Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Decision support for migrating application's functionality to FaaS

Rawad Hamzeh

Course of Study:	Computer Science
Examiner:	Prof. Dr. Dr. h. c. Frank Leymann
Supervisor:	M.Sc. Vladimir Yussupov
Commenced:	November 16, 2018
Completed:	June 6, 2019

Abstract

We live in the era of cloud computing. Many companies move their legacy applications to cloud environments for the sake of improving their business and achieving more profitable success. Choosing the cloud service model is one essential step in the process of transition applications to a cloud environment due to the importance of its contribution to optimizing the applications performance and costs.

The variety of cloud service models bring along different aspects in terms of both user responsibilities and cost efficiency. Function as a Service (FaaS) is one of those models that looks appealing and convenient for users. This model provides users the ability to move parts of their applications to the cloud environment. The FaaS environment is responsible for managing, initializing and scaling the functions. Furthermore, the user is billed based on actual usage of the infrastructure. However, the interesting question is how one can choose the functions that fit the FaaS model, and what are the factors that could influence the selecting process.

In this work, we introduce a new approach to help developers and software architects in assessing the suitability of application functions for running in FaaS environment. This approach considers the factors that might influence the appropriateness of function for being deployed in FaaS model and provides users with a score representing the suitability of function for FaaS environment. We validate our approach by means of a prototypical implementation. This new approach does not consider the security concerns in the assessment process. But, this work could be considered as a starting point for automating the FaaS functions selection process.

Acknowledgement

First, I would like to thank Professor Frank Leymann for the opportunity to work on this thesis at the Institute of Architecture of Application Systems, University of Stuttgart.

I would like to thank my supervisor M.Sc. Vladimir Yussupov at the University of Stuttgart for his continuous support, insightful comments, motivation, and feedback during the progress of my master's thesis.

Besides, I would like to dedicate this work to the soul of my late father Bayan, and to the soul of my late grandfather Fawaz. You have been always my motivation and the source of my strength. Also, I would like to express my great profound gratefulness to my mother Eitaf and grandmother Shukria for their continuous love, motivation, support, and encouragement. I am very grateful to my beloved sister and the secret of my happiness Roaa for her encouragement and love.

Lastly, I am thankful for all my family member who supports me to achieve my dream.

Contents

List of Figures

1 Introduction

In this chapter, we focus on the concepts underlying the topic of research. We present the motivation behind the topic and describe the problems needed to be solved. Then we define the primary research questions of this thesis. Finally, the general structure of this work is presented.

1.1 Research Motivation

One of the main factors for achieving business efficiency and improving competency is the employment of cloud computing. Cloud computing refers to the concepts and techniques for providing on-demand computing services with high reliability, scalability, and availability in a distributed environment. The National Institute of Standards and Technology (NIST) defined cloud computing as: *"(...)a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."* [MG+11]

Using cloud-based services has become a common avenue for leveraging remote computing resources, with the benefits that include reduced costs, increased automation, greater flexibility, and enhanced mobility [KT14]. Furthermore, when the same service is used by multiple clients, a cloud service provider can more effectively improve quality and reduce costs by leveraging the economies of scale. And from the service maintenance perspective, replacing a locally implemented functionality with a remotely maintained service might reduce the maintenance burden and facilitate functionality update process. With the rapid evolution of IT solutions, computing technologies have improved enough to answer business demand at the desired speed. Nevertheless, it is still challenging for business users and programmers to apply and use computing capabilities in the right way due to their limited knowledge of technology and how to change a legacy system to cloud computing with selecting properly deployment model. In spite of all the advantages of using cloud-based services, moving a system or part of a system to the cloud environment to effectively leverage the benefits of cloud resources requires changes to the application's source code [KT14]. These changes might be costly and not promise to make a change in performance. However, assessing the suitability of such code for moving to a cloud environment is cumbersome to be performed manually by the software engineer. Hence, the software engineer needs help in performing such an assessing process. More specifically, a code assessing process, which guides and helps the software engineer, is required in order to ease the process of checking the suitability of function to move into a cloud environment.

In this thesis, our motivation is to bridge the gap between a software engineer's perspective and the technical side of cloud computing by benefiting from revolutionary paradigms that reducing server management efforts. More precisely, our goal is to find a new approach that performs a suitability assessment of a system function for deploying in a cloud environment. Furthermore, since the main issue, as described before, lies in the difficulty of making such checking manually by programmers and software engineers. Hence, this new approach will be focusing on reducing the efforts and skills needed from users for the sake of taking the right decision regarding moving code to a cloud environment.

Cloud computing paradigm becomes a success story as new useful technology across the globe. Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) are the three major cloud service models. Each one of these service model has its effects and reaches to various industries[WLY+10]. Generally, the architectures of the applications change to be more depended on containers and microservices. Thus, Serverless computing emerges as a new compelling paradigm for cloud applications. More precisely, serverless computing is a paradigm that allows running the system server-side code without addressing the server management issues. As a part of this paradigm, FaaS concept appears to allow programmers deploying small pieces of code in the structure of a function that runs in a cloud environment.

The frequent changes in business demands, the increasing need for de-centralized methods, encourage the software architects to move the current systems to the cloud environment and leverage from its benefits regarding the availability and scalability. Therefore, taking apart of the code and deploy it to the serverless platform might help to face those challenges. But we should realize that changing the code is difficult, costly, and error-prone ways [KT14]. Moreover, selecting the appropriate functions that are worthy to move them to a serverless environment in order to make a diffecrence in performance is not trivial. Bridging this gap is crucial for achieving operational excellence and efficiency, and it is getting more complicated with the fast evolution of cloud technologies and the increasing complexity of the functions. The following section will expand deeply our research problem and the resulting questions.

1.2 Research Problem

Despite the tremendous progress that has been made in cloud computing field, programmers are still not able to thoroughly defining which functions or pieces of code are suitable for the serverless environment and can work efficiently in the FaaS model. Furthermore, moving to the serverless computing and FaaS model requires the programmers to have adequate knowledge of how to select suitable functions' code for FaaS environment. To do that, programmers might depend on cloud computing experts who are equipped with the domain knowledge that the programmers already know, and they should be aware of how to decide about the suitability of serverless computing. Thus, our research problem is described as the following:

Problem Description: *The major question in bridging the gap between existing systems(legacy code) and serverless technology is how to build a model that encompasses the concepts and the rules of the FaaS model and analyze the source code of system functions with building a profile for each function in the context of suitability for FaaS environment. Consecutively, is how to empower software engineers to use this model for managing the whole process of selecting suitable serverless functions, by building a score that can be interpreting the suitability of system function to be moved into the serverless environment and conducting the results to the user as useful insights.*

To focus on this problem, we analyzed it into a set of related research questions that we listed in the following section.

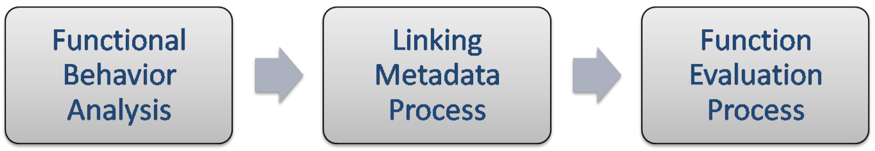


Figure 1.1: Abstract Process I

1.3 Research Questions

With respect to the problem description in the preceding section, we define a set of research questions that are induced from an initial inquiry:

How to design and build a recommendation system for providing insights about functions suitability for serverless computing based on the analytics data and best practice use cases?

This problem could be decomposed into the following set of research questions:

- How to analyze functions' source code of existing systems and decompose them into metrics?
- How the behavior of the function might influence its suitability for running in FaaS model?
- How to automatically interpret the results of the analytics features, and generate the appropriate recommendation score that identifies the suitability of the functions on the serverless model?

1.4 Research Scope

The research scope is limited to investigating the use of functional behavior and functions syntax suitability in order to facilitate the process of assessing functions and checks their suitability for cloud environment. The work includes designing the system architecture and validating it through a prototype implementation that demonstrates the viability of our approach. The functional behavior analysis is projecting the software source code into a set of a predefined function-behavior-structure and thus into the broader framework of engineering design. By doing so, we can draw some expectations about the state and behavior of serverless functions. The linking meta-data process gathers the measurable data about the system functions regarding the behavior as well as the context of calling the functions. Lastly, The function evaluation process is responsible for producing a score representing the suitability of the function for being moved into the FaaS environment. In Figure 1.1, we show an abstract process of our approach.

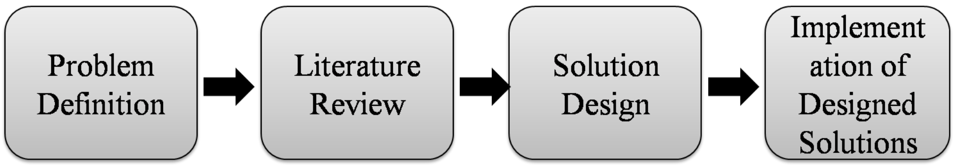


Figure 1.2: Research Process

1.5 Research Contributions

The contribution of our thesis can be described as follows:

- In this thesis, we introduce a new approach to providing programmers and software architecture with needed insights about the suitability of system function to be deployed as FasS model without getting into technical details or having deep knowledge about the required analytics capabilities to answer their insights. This method might give a good opportunity for users to get the right insights and decide if it's worthy to move the function to the cloud model without much investing;
- We also introduce a new way and concept to align function analytics with meta-data as well as business goals and apply efficient mechanisms to automatically profile the function and decide how much it is appropriate for transition to a serverless computing environment in a flexible and easy manner;
- Our proposed approach is supported by a prototype implementation that involves a mixture of behavior analysis and syntax analysis that is empowered by meta-data.

1.6 Research Methodology

In this thesis, we followed the methodology proposed by (Hevner et al. 2004) for conducting research in design science. Design science refers to a systematic way of creating a problem-solving process. This methodology has proved its efficiency in rationally decomposing research problems, deriving the research process to satisfy the needs of a particular domain, and providing additional value for future research to build on. In Figure 1.2, we explain our research process that consists of several steps and iterations in order to ensure delivering quality and value in results. Those steps are as the following:

- **Problem Definition:** Define the research motivation, research problem and the derived questions, the research scope and expected contributions;
- **Literature Review:** Conduct a review of recent work in Serverless computing and FaaS model;
- **Solution Design:** Introduce a concept and design of the proposed approach;

- **Implementation of Designed Solutions:** Implement a prototype recommendation system based on our previous findings.

1.7 Report Outline

This thesis report is organized into five chapters. The first chapter consists of a general introduction to the topic, followed by the research motivation and the research problem, then we define the research questions and the followed methodology to answer them. The second chapter is dedicated to present a background knowledge around the topic and a state of the art about the relevant concepts and techniques. In the third chapter, we will present the concept and design of the proposed solution. The fourth chapter covers the implementation and its validation. Finally, in the fifth chapter we discuss the conclusions, limitation and the future work.

2 Background and Literature Review

This chapter is dedicated to present a state of the art about the main topic of the thesis. We are going to discuss the notions and the evolution of the methodologies and techniques in the related fields such as cloud service models, sereverless computing paradim, function code analysis and profiling approach. The primary purpose is to focus on the concepts underlying our research work, discuss those basic concepts and review the relevant scientific work.

2.1 Cloud Computing Service Models

Cloud computing is the delivery of on-demand computing services typically over the internet and on a pay-as-you-go basis. And the main idea behind this concept is sharing computing resources, and it is an alternative to having local servers handle applications. Cloud computing groups together large numbers of compute servers and other resources and typically offers their combined capacity on an on-demand, pay-per-cycle basis. Rather than owning their own computing infrastructure or data centers, the end users of a cloud computing network usually have no idea where the servers are physically located, they just spin up their application and start working. According to [WVZX10], Cloud computing is *a natural evolution for data and computation centers with automated systems management, workload balancing, and virtualization technologies.*
Moreover, one of the fundamental concepts behind cloud computing is that the location of the service, and many of the details such as the hardware or operating system on which it is running, are hidden from the user. Currently, Cloud computing services cover a vast range of options, from the basics of storage, networking, and processing power through to natural language processing and artificial intelligence as well as standard office applications. Mainly, there are two primary deployment models of cloud: public and private. Most organizations use a combination of private computing resources (data centers and private clouds) and public services as a hybrid environment. Actually, cloud Computing is considered a good leap in the computer science world as it has many benefits, basically:

- Cloud technology is paid incrementally, saving organizations money
- Elastic nature of the infrastructure to rapidly allocate and de-allocate massively scalable resources to business services on a demand basis.

Cloud computing is typically divided into three levels of service models: Infrastructure as a service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The Figure 2.1 clarifies the relation between these models and the required management level.

The kind of cloud service models that software architects should consider depends on required particular performance, security requirements, and their specific business goals [IBM]

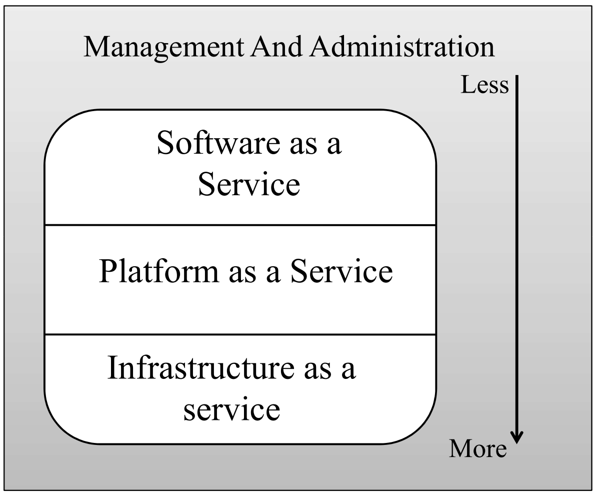


Figure 2.1: Delivery Models [HBKH09]

2.1.1 Infrastructure as a Service(IaaS)

Infrastructure as a Service (IaaS) is the delivery of hardware (server, storage and network), and associated software (operating systems virtualization technology, file system) as a service. It is a form of hosting. It includes network access, routing services and storage. It may also include the delivery of operating systems and virtualization technology to manage the resources [HBKH09].

The IaaS provider will generally provide the hardware and administrative services needed to store applications and platforms for running applications. Additionally, scaling of bandwidth, memory, and storage are generally included as IaaS features since the service provider owns its hardware equipment and he is responsible for hosting the systems. Instead of buying and installing application in local data center, the IaaS customer rents computing resources and the user is billed depend on his usage of the computing resources [BJJ10]. However, most users consider the key benefit of IaaS to be the flexibility of the pricing, since we should only need to pay for the resources that our application requires. More companies are looking to save costs and gain flexibility by leveraging the infrastructure that can be used on demand.

To be more precise, IaaS provides an environment for running user virtualized systems in the cloud, and virtualization techniques are often used with this cloud service model as well. Virtualization separates resources and services from the underlying physical delivery environment. Therefore, with this approach, the user can create many virtual systems within a single physical system. Once the virtual machine is started the IaaS environment can ensure that the running virtual machine continues to look healthy as a whole. The computers needed to run the application and the raw storage that is needed by the application are owned and supported by the IaaS environment [DTM10]. Thus, the IaaS introduces the *dynamic scaling* feature which means that the infrastructure can be automatically scaled up or down, based on the requirements of the application.
Here some characteristics of using IaaS [HBKH09]

- Computing resources are available as a service;

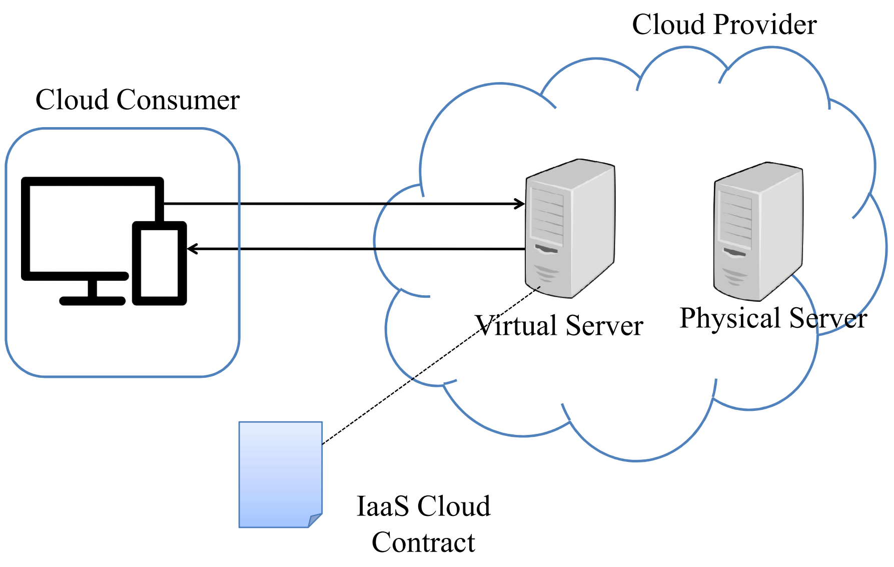


Figure 2.2: IaaS Model [Ibr]

- Dynamic scaling,Infrastructure is scalable depending on processing and storage needs;
- Self-service provisioning,which enables the user to obtain resources through a self-service without relying on IT to provision resources;
- Desktop virtualization;
- Saves enterprises the costs of buying and maintaining their own hardware.

One of the main advantages of IaaS, it gives clients complete control of their infrastructure. Furthermore, IaaS is an option that is very flexible and is a good choice for moving applications to the cloud when there is no time to rewrite the application's code for other cloud delivery models. IaaS is also suitable for large companies that want to have complete control over their applications and infrastructures, but are looking to only purchase what is actually needed. For rapidly growing companies, IaaS can be a good option since they don't have to commit to a specific hardware or software as their needs change and evolve. Despite all the advantages of IaaS, there are a lot of downsides, we'll mention some of them, like the companies, are responsible for the upgrades of software developed by themselves and security might be an issue because security features of the IaaS Cloud provider may not adequate for your needs. In Figure 2.2 we can see a cloud consumer using a virtual server within a physical server in the IaaS model.

2.1.2 Platform as a Service(PaaS)

Platform as a Service(PaaS) is a cloud computing service model that provides users with a cloud environment in which they can develop, manage and deliver applications. In addition to what IaaS offers, it provides prebuilt tool that enable users to develop, customize and test their own applications. In other words, PaaS consists of IaaS plus a certain amount of application software which help the user to build his application [HBKH09]. Platform as a Service (PaaS) can be considered as an application development and deployment platform delivered as a service to developers over the Web. Therefore, PaaS is defined as a service that provides an abstracted and integrated environment for the development, running, and management of applications [IBM].
It facilitates the development and deployment of applications without the cost of buying and managing the underlying infrastructure. Moreover, PaaS model is providing all of the facilities required to support the complete life cycle of building and delivering web applications and services. Typically, these software services include a database, middleware and development tool.
However, the key benefit of a PaaS environment is that users don't have to be concerned with some of the lower-level details of the environment, we mean there's no management skills or maintenance efforts are required for managing the infrastructure. Every stage of the software development process, from the design stage onward, lives in the cloud. The main goal of the PaaS model is to create an abstracted and repeatable process for the creation and deployment of high-quality applications [BJJ10]. Although the user has no control over the fundamental infrastructure like storage and other hardware, they have control over the kind of end user application deployed to them.

Furthermore, the PaaS model supports hosting services that can be used by multi-users. typically, it addresses the need to scale as well as the need to separate concerns of access and data security for its customers and thus, it supports the dynamic scaling feature. In PaaS context, the dynamic scaling means that the software can be automatically scaled up or down.
Here some characteristics of using PaaS [HBKH09]:

- Provides a variety of services to assist with the development, testing, and deployment of apps;
- Eliminating the installation and operational burden from companies, they is the responsibility of PaaS model;
- Numerous users can access the same development application(multi-tenants);
- PaaS is built on virtualization technology, meaning resources can easily be scaled up or down as your business changes;
- Enables programmers to focus on development without having to worry about underlying infrastructure.

Concerning the suitability of using the PaaS model, it is a good choice if the company wants to be able to create its own customized applications without the worrying about maintaining the software. This cloud service also can greatly reduce costs and it can simplify some challenges that come up if users rapidly developing or deploying an application. Moreover, it reduces the amount of coding and makes the development and deployment process of applications simple and cost-effective.
In spite of the fact that PaaS model has many benefits for users, it has some drawback sides. The fundamental disadvantage of Platform as a Service is users might be got locked into a particular vendor or environment and it will be very costly and time-consuming to change to another vendor. In addition, data security needs to be considered because the data might be stored in shared cloud

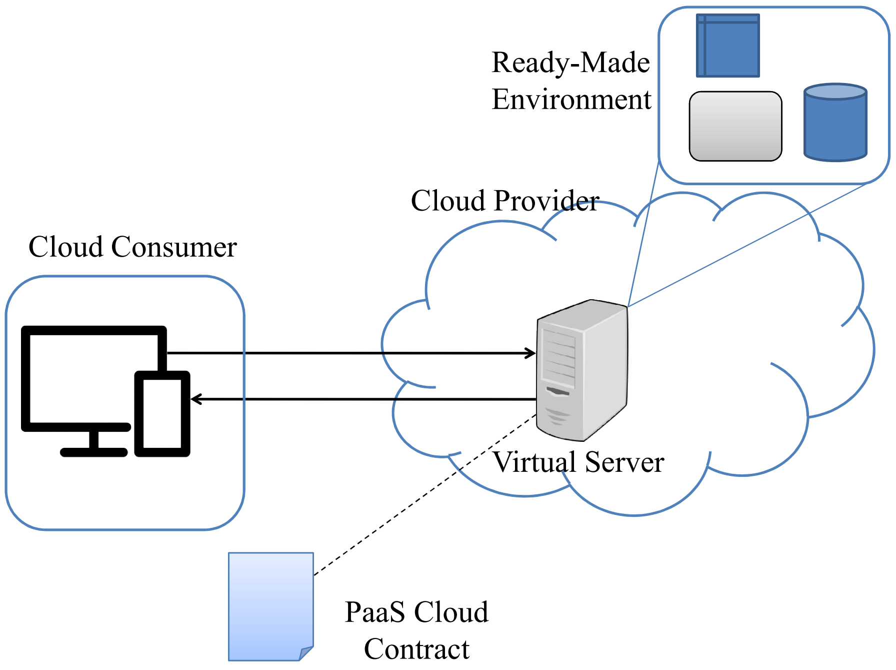


Figure 2.3: PaaS Model [ore]

services, therefore, users should address this issue and keep confidential information out of the cloud. In Figure 2.3 we can see a cloud consumer using a virtual server and software applications in the PaaS model.

2.1.3 Software as a Service (SaaS)

Software as a service (SaaS) is a software distribution model in which a third-party provider hosts applications and makes them available to customers over the Internet. The SaaS application sits on top of both a PaaS and foundational IaaS [SFL11]. Software as a Service solutions are operated and maintained by the software vendor itself (Xinand Levina 2008)[XL08]. The vendor takes over activities that previously lived within the responsibility of the customer and extends its area of responsibility beyond development activities to operating and maintaining. In other words, users do not have to install applications on their local servers. Instead, the applications reside on a remote cloud network accessed through an API. Depending on those applications, users can store, analyze data and collaborate on projects. In Figure 2.4 we can see a cloud consumer using a software service within a cloud server in SaaS model.

SaaS uses one single instance of a software application running on top of a multi-tenancy platform and provides delivering software functionalities to a large number of users. Users do not have to install software packages in their local machines or environment. Instead, they subscribe to desirable service using the SaaS model. Moreover, users can customize the service or software with their unique business needs through easy configuration and customization. Afterward, they log into SaaS and act as a consumer of the SaaS services over the web [SZG+08].

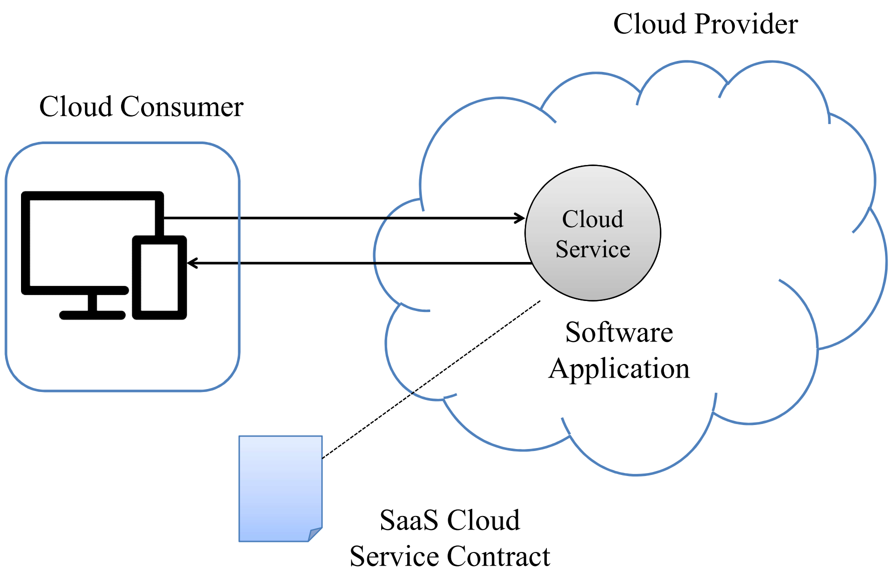


Figure 2.4: SaaS Model [pac]

The key advantage of Software-as-a-Service model that business companies get the immediate benefit of reducing their development expenditures. In addition, a business gains the flexibility to test new software on a rental basis and they can continue to use and adopt the software if it meets their business need[SZG+08].
Here some characteristics of using SaaS [HBKH09]:

- Applications are accessible from almost any internet-connected device, from virtually anywhere in the world;
- Users do not have to manage, install or upgrade software; SaaS delivery model manage all of them.
- Dynamic scaling in term of SaaS; which means the usage of resources can be scaled according to services needs and business plan;
- Flexible payments, where users pay for this service on a monthly basis using a pay-as-you-go model. Saving cost from the development and deployment process might allow many businesses to exercise better and more predictable budgeting.

In addition to the previous characteristics, SaaS is considered an affordable solution for start-up companies that want to launch their services quickly and don't have time for solving and managing infrastructure or software issues, where it reduces the time and money spent on boring tasks such as installing, managing and upgrading software. On the other hand, it gives the users and programmers plenty of time to spend on solving more pressing matters and issues within the company.
Despite all these tremendous benefits, users must rely on outside vendors to provide the software,

keep that software up and running, which means users turn much of their control over to a third party provider. Providers that experience a security breach or any other issue can have a profound effect on the customers' ability to use those SaaS offerings. However, there are still many software applications that don't offer a hosted platform in a SaaS environment. User may find it necessary to still host certain applications on site [XL08].

2.1.4 Function as a Service (FaaS)

FaaS is a new way of building and deploying server-side software, oriented around deploying individual functions or operations to the cloud environment. It enables developers with little to no experience of operational logic to create, monitor, and invoke cloud functions [EIST17]. In simple words, FaaS allows developers and programmers to move a small piece of source code to the cloud environment in order to benefit from the cloud computing features like managing and auto-scaling. This code called a cloud function and is responding to a variety of events. Once an event happens, the cloud function runs in the cloud environment and achieves its task [Win].
Even though the FaaS model is considered in top of the hierarchy abstractional management level from the user perspective, the FaaS user can still make his own configuration and modify parameters of the running environment such as memory size or number of CPUs of the underlying function host which influence the operation of the deployed cloud function [EIST17].
Figure 2.5 shows the proposed layered model for FaaS, the first layer called the operational logic layer which responsible for deploying, executing and monitoring the function code. This layer could include routing and throttling incoming requests to functions, isolating functions using virtualization, provisioning (auto-scaling), and allocating appropriate resources to the functions. The second layer is called the function/business logic, which represents the function logic to be run itself. Lastly, the event integration layer is responsible to define when, why and how the function code should be executed. It is considered as an integration layer because it couples the function logic with events sources and listens to the occurrence of arbitrary events in order to trigger the cloud function as a response to an event. These events can originate from timers to execute the cloud function periodically, from incoming messages, from message queues, from changes to files and other system objects, or from HTTP requests to a specific endpoint [EIST17].
The main Idea in FaaS is addressed as follow: the same function logic is coded once and deployed to the cloud environment in order to be used by the users. Hence the cloud function is coupled to some source events that responsible to trigger the cloud function. In Figure 2.6 we see an overview of how the FaaS model works and what are the main components. These components can be categorized to

- *Event sources* which trigger one or more cloud function instance;
- *FaaS Controller* that play the role of managing and controlling the function instances. Furthermore, it tyes the function with its event sources;
- *Function instances*; a single function logic, that can be scaled with demand.
- *Platform services*; the underlying infrastructure required to keep the environment running.

In order to understand the mechanism of FaaS model, we will introduce the lifecycle of function that will be deployed to FaaS. It starts from writing code by a programmer and enriches it with metadata and specifications, a builder entity comes to combine both the source code with metadata,

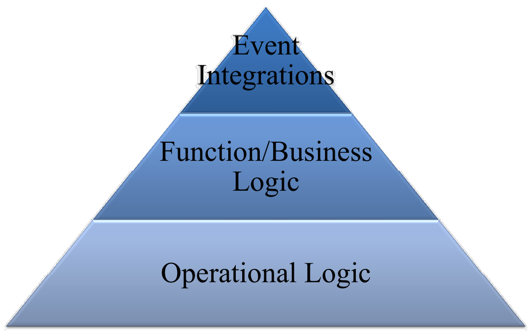


Figure 2.5: The FaaS Model of Cloud Computing [EIST17]

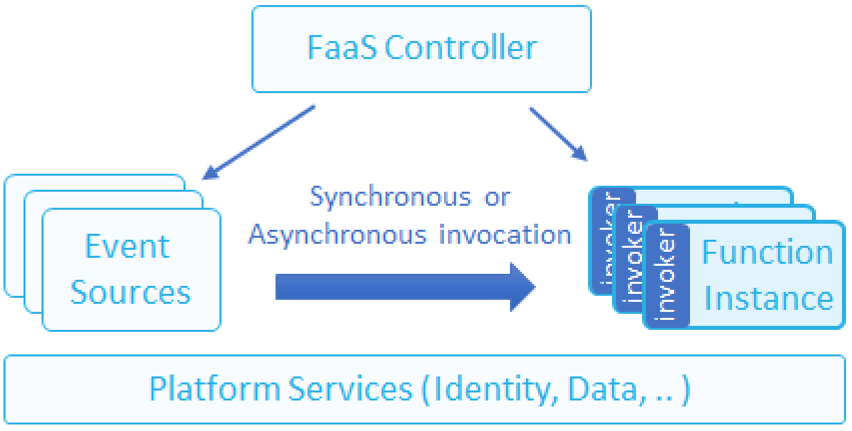

FaaS Controller
Synchronous or Asynchronous invocation
Event Sources
invoker
invoker
invoker
Function Instance
Platform Services (Identity, Data, ..)

Figure 2.6: The FaaS Main Components [Win]

compiles them and turn them into a package that can be deployed to a cloud server. Hence the controller component in the FaaS platform manages the scaling, which means the number of functions instances depend on the events traffic and/or load on the instances [Win]. Figure 2.9 shows the life cycle of a function in the FaaS model.

Furthermore, the FaaS delivery model supports two main functionality features. First, it supports the event streaming, therefore the processing might need queues in order to ensure a response to every event. On the other hand, the function that has already deployed in the FaaS environment has a warm startup since the function is already deployed and is ready to serve the event which means the function has a minimal number of instances at any time, such that the "firstëvent received has a warm start.
In general, developers in the FaaS model should focus on systems and applications composed of event-driven functions that can respond to a set of triggers, the FaaS platform is responsible for managing and control the cloud function. The main advantage of such a mechanism, there is no

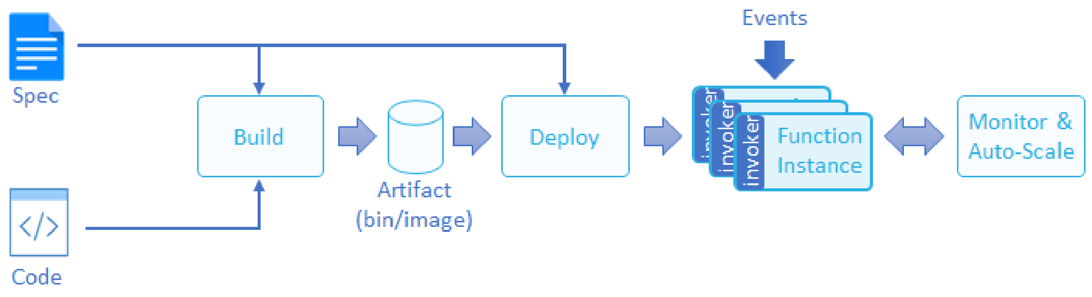


Figure 2.7: The life cycle of a function in the FaaS model [Win]

need for underlying infrastructure management like the operating system or runtime management. FaaS takes care of all the management issues from automated scaling and decides how many instances of the function should be initialized, elastic load balancing and "pay-as-you-go"computing model[EIST17].
FaaS is addressed to be a suitable solution when the companies want to focus on business logic within the system function and those functions should respond to the increasing demand on them by having the ability to automatically scale up and down. Furthermore, FaaS model meets the desire of companies in tieing the cost with the actual transaction. Therefore, developers creating event-driven applications, such as those that respond to database changes, IoT readings, human input, etc.

FaaS Benefits

Actually, there are many advantages from using FaaS model, we will mention some of them [EIST17][Win]

- Appropriate model for event-driven applications with unpredictable workloads especially with the emerging topics IoT, data, messages;
- Programmers can focus more precisely to business and function logic rather than take care of operational issues which become FaaS platform concerns;
- Meet the desire of business users that only pay for the actual time of the function code is running which called a new "pay-as-you-go"cost model;
- Clear separation between business and operational Logic, as the operational logic becomes within the responsibility of the cloud platform, whereas business logic takes all the concerns of the programmers, and as a reflection of that, companies require to clarify the roles of Dev and Ops in their team;
- FaaS function will run for a short period of time.
- The time needed to initialize the function and be ready to run is very short compared with other models.

FaaS Drawbacks

Although all the previous advantages of using FaaS mode, there are still some challenges might face users:

- The time needed to initialize an instance and be ready to run if the FaaS environment had removed all the instances of function when it was in the Idl mood;
- Debugging might be an issue that faces developers because in such a dynamic runtime environment it becomes more complicated to track and debug the running code;
- Not suitable for running long-running business processes since these function (services) instances will destroy after a fixed time duration;
- Potential for platform lock-in due to the platform's programming model, eventing/message interface;
- Since server instances are come and go, maintaining the state of an application is really challenging with these types of frameworks;
- End-to-end testing or integration testing is not easy with the functions come and go;
- It is still suffering from a lack of standardization maturity;
- Migrating legacy systems, how to optimize the migration process, and to what extend it is possible to automate the extraction of functions from legacy systems to cloud functions.

Function Requirements

In the sake of approaching to an effective and efficient performance when moving to FaaS environment. There are some rules that make functions more appropriate for the transition to FaaS, in principle, we can re-architect any function to suites the FaaS platform which might be costly some times. However, the function logic is preferred to have some characteristics that fit the FaaS platform. Firstly, we should have a clear separation between the operational logic of the function and the underlying implementation of its depended event classes. Second, the connections streams used by the function are preferred to stay for last-longing because it makes the execution faster than re-initialize and re-opens such connection streams. Additionally, the event-driven concept is the main core concept that drives the FaaS function, where events sources will be the triggers to running the functions. Such triggers might be a new modification in a database, new messages received or an invocation from another function. Accordingly, the function logic should be built in a way that can be listning to some events sources in order to be triggered and start running [Joh17]. Moreover, the function might be configured to be triggered by a set of events together, and an event could trigger multiple functions as well.

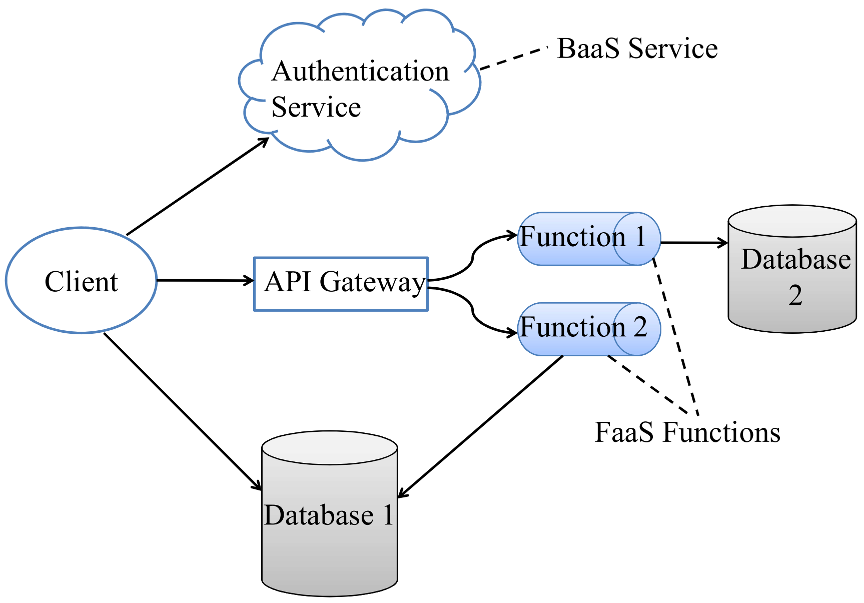


Figure 2.8: Example of ServerLess FaaS Model and BaaS Model [Rob18]

2.2 ServerLess vs. Function As a Service

In this section, we'll present what the concept serverless means and what is the difference between FaaS and serverless, and then we'll show some details about the FaaS in this context. Serverless, despite its confusing name, is a style of architecture where we rely on running our own server-side systems as part of our applications to a smaller extent than usual [Joh17]. We do this through two techniques:

1. BaaS, where we tightly integrate third-party remote application services directly into the frontend of our apps, and
2. FaaS, which moves server-side code from long-running components to ephemeral function instances.

Backend as a Service (BaaS): BaaS services are domain-generic remote components that we are able to integrate into our application or product by an API. In other words, it is about using third-party services (like provider services) in building our server side components instead of coding them by ourselves. Simply, users break up the application into small parts and implement some of those parts using external services. And here the main difference between SaaS (see Section 2.1.3) and BaaS, whereas SaaS concerns about outsourcing the whole business processes.
In BaaS, users depend on application logic that other third-party has implemented and deployed it, more precisely they tightly integrate those remote application services directly into the frontend of the designed application. An authentication service is an appropriate example for BaaS whereas this piece of logic is similar across many application. instead of reimplementing it again for each

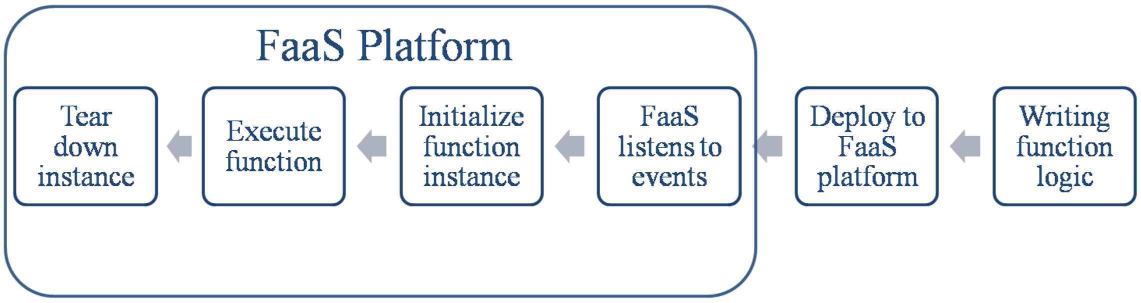


Figure 2.9: FaaS Functions life cycle

application, developers can integrate their application with such remote service [Joh17]. The Figure 2.8 show us a brief idea of how the difference between FaaS and BaaS.
Function as Service (FaaS) is the second technique of serverless concept which we describe it briefly in Section 2.1.4, and here we will complete some issues regarding it.
Similarly to what we mentioned before, FaaS is a new approach that focuses on just the individual operations or functions that express our application's logic, and concerns deploying the functions' logic or operations to a cloud environment in order to build and deploy the server-side software of our application.
The life cycle of the FaaS function is different from the function life cycle in traditional systems, see Figure 2.9. In the FaaS platform, the function is not active at all after it has deployed by the user, accordingly, the platform starts listening to events that specified for each function. Once a relative event occurs, the platform initializes an instance of the respective function and call the instance with the triggering event. Once the function has finished executing, the FaaS platform is free to tear it down. Alternatively, as an optimization, it may keep the function instance around for a little while until there's another event to be processed [Joh17]. In contrast to a traditional system where the function always in idle status.
Consequently, FaaS can be considered as inherently an event-driven approach. Accordingly, FaaS platform can incorporate with event sources in both synchronous and asynchronous form. For integrating with asynchronous event sources, it uses message bus or object store. Whereas, API gateway can be used for integrating with event source in the synchronous form. Hence, the API gateway in FaaS platform receives requests from users' client and routes them to the respective FaaS functions which can be considered as backend handler, afterward when the handler returns the result, it takes the response back from the function and finally returns the response to the original client [Joh17].
One of the significant characteristics of FaaS functions is stateless functions which means that the state of the function might be got lost across multiple invocations. More precisely, there is no such guarantee that a state from one invocation of function will be persistent to another invocation of the

same function. Consequently, if the function is required to be persistent in term of state, then the state should be externalized and stored outside of the function instance. This issue is considered one big change as users no longer store any session state within the server side application code, and instead persist all of it to the external store [Rob18]. However, the approach of this mechanism has a significant impact on scaling, as it lets the scaling feature to be more efficient [Joh17]. On the other hand, this approach could be a new challenge to the users because it could have a large impact on the application architecture, and software architects might have to change the architecture of the application in order to make the FaaS function stateless.
Accordingly, it is worthy to mention very briefly the key difference between FaaS and PaaS and how to choose between them. In PaaS the developer should think about the scaling and how to manage the size and shape of your clusters based on on-demand requests. Whereas, FaaS is much more efficient when it comes to costs. As the scaling in FaaS is completely transparent. However, programmers is still using PaaS when they have synchronous-request–driven components with many entry points. In other hand, FaaS is considered better choice for asynchronous event-driven style with few event types [Rob18]
According to previous, Faas and BaaS are very different from each other. In FaaS we are thinking of moving individual functions code from our application to the new cloud environment. Whereas, BaaS concerns in moving whole parts of the application to the cloud by using third-party services. Nevertheless, the common key between both is that users don't have to manage the server process, instead the cloud environment does it. Furthermore, in both the application components are separated which allows the programmers to add new functions or switch the logic of existing ones without the need to re- arcature the system. Scaling, high availability, and security are also qualities that are common in FaaS and Baas. In addition, both succeeded in reducing the time of the developent process and implementation phase sharply and noticeably which reflects that to the whole development process and let it faster [Rob18][mar].

Main advantages of using ServerLess:

1. The total operational cost is reduced noticeably on both sides:
    - Companies don't have to spend lots of time in implementing and managing the server which reduces the labor cost;
    - Companies rent the infrastructure from the cloud and pay exactly based on what resources do they use.
2. Optimization: when programmers optimize the performance of the application, through code or a change in architecture, they'll automatically see cost savings without having to change anything about their deployment. in other words, any performance optimizations are made to function code will not only increase the speed of the system, but they'll have a direct and immediate link to a reduction in operational costs;
3. Self auto-scales: serverless increases the flexibility of scaling compared to the other cloud service models, as scaling happens automatically with no effort. and this feature is considered the core of all other features because auto-scaling allows programmers to pay only for the computing resources that needed and actually used;

4. Shorter lead time: serverless moves a lot of complexity of the building, deploying and operating applications to the cloud environment and therefore it speeds the way of delivering a software product and leads to shorter time that helps companies making innovations.

Main limitation of ServerLess:

1. Lose up some control of the system and a vendor takes over the controls, then he might make some enforcement to users such forced API upgrade, changing in cost and so on;
2. Loss of server optimizations when using BaaS: With a full BaaS architecture there is no opportunity to optimize the server-side design. Because in BaaS, users use third-party services which they don't have the right access to change the process inside them;
3. Serverless FaaS works with stateless function: there is no guarantee to persist the state of instance from on invocation to other invocations, as the platform keeps the instance for a specific limitation of time and if no such event happens then it kills the instance. The only way to persist the state is to externalize the state to external DB.

2.3 The Pricing Model of Serverless-FaaS

Understanding the general concept of a pricing model used in FaaS serverless is a fundamental affair in the sake of taking the right decision regarding transfer a function to FaaS serverless environment. In spite of some tiny variation between FaaS vendor in pricing the FaaS function, however, they use the same approach of calculating the cost of running FaaS function. The pay-per-use pricing model is the common model used for this purpose. In this model, users pay for what really use, and the pricing calculation is based on three factors [ama]:

- The number of function invocations;
- The function execution time, it's from the time the code begins executing until it returns or terminates;
- The usage amount of computing resources, so the price depends on the amount of memory the function allocates, CPU power and other resources.

This kind of model increases the satisfaction of users, where users pay for exactly what they use. Moreover, the flexibility in deploying to FaaS environment encourages users to move their application to FaaS. Basically, users write their own functions and simply deploy them to FaaS environment which is responsible for all the underlying server management. Typically in the FaaS model, there are no resources allocated or chargeable until a function is called. in other words, we can say that the FaaS function is just a block of code that can be triggered on demand by predefined source events but doesn't use any run-time resources such as CPU or memory until then.

Despite all the attractive features coming from using FaaS environment, however, users should plan carefully once they decided to move their application to it. Because in some cases using FaaS might be tricky regarding the cost if there's no accurate planning. since having a reserved server instance based on the IaaS model would be cheaper than having a function running continuously for the

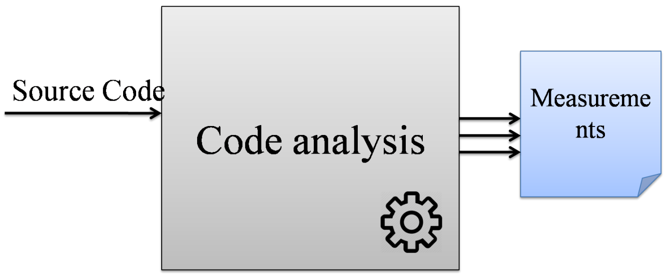


Figure 2.10: Code Analysis

entire month. In fact, FaaS is not the best choice for continuously compute-intensive workloads where some serious processing power is needed to manipulate through a lot of data.
Therefore, software architects and operational designers need to consider the functional and non-functional requirements of their business needs and review carefully the pricing details supplied by the cloud provider. Because in some cases when the hits of the function increased sharply, the cost of the FaaS function would be doubled compared to reserved compute instances based on IaaS for example. another case if the FaaS function invocation failed, it will be retried for as long as the event is retained which leads to some substantial overhead. Hence, It's important to monitor the costs regularly or have automatic checks [Eiv17].
Eventually, the economic benefits of FaaS computing heavily depend on the execution behavior and volumes of the application workloads. consequently, it's so critical to set an accurate cost plan and evaluate it based on business needs. Nevertheless, the cost of running software systems is not the only reason to consider FaaS offerings. Building solutions based on small and focused units of business logic that can be quickly and cheaply delivered to the market and effortlessly managed and scaled based on the actual demand offers a massive competitive advantage in the marketplace.

2.4 Source Code Analysis

In this section, we show a brief idea about the source code analysis in general and what is the purpose of using it.
Code analyzer is a program that takes other programs as an input, scans their source code, finds out some important metrics and properties and returns reports on the source code structure of a program or software. Moreover, it is used to find bugs and faults that may not be obvious to a programmer, in order to fix them before the application is being used for its purpose. Typically, a code analyzer is designed to review source code for known errors, programming context, and overall program structure like code complexity and code clearness.

Source code analysis is considered as a static code analysis, where the source code is analyzed simply as text source code even though the program is not running. Whereas there are a dynamic code analysis concerns in monitoring the behavior of source code during runtime and find any deviation in executing the source code as well as taking critical measurements and metrics that are required by the software architects. In addition, the logic layer in static code analysis is applying different algorithms based on what the users are looking for. For example, find the code complexity or the number of depended classes. The results of such algorithms might help users to understand the code deeply, discover any fault in the logic of the code, finds specifics codes pattern inside the source code and discover new ones. Hence, the precision in the result can be configured by business users depending on their need as high precision might affect the performance and take a longer time, whereas lower precision would be faster [MS18].

On the other hand, static code analysis is used in the software testing process, as it is considered a part of *white box testing approach*, where it ensures and validates the internal framework and components of a software application as well as verifies the source code according to design specifications, which means the source code is accessible by the testing engineers. In contrast to the *black box testing approach* where it focuses on the analysis of software functionality, checks if it meets the client requirements and software specifications without having access to the source code. Indeed, the static code analysis is involved in many software testing approaches due to its efficiency in detecting faults and bugs early. Mainly, this analysis is used by software testers, quality assurance staff, and developers in the sake of improving the overall code structure and facilitating code optimization.

2.5 Related Work

Improving the performance, reliability, and scalability of software applications is a main concern to clients and software developers.Using cloud-based services is an essential concept that fulfills this need and contributes in enhancing software applications.However, transforming an existing application to use cloud-based services is error-prone and not an easy process to perform. Therefore, many researches and works were focused on different area of using these cloud-based services and transitioning the existing applications. Some work focused on helping the programmer to evaluate the overall performance of cloud functions taking into accounts the variety and heterogeneity of these functions, which is described in Section 2.5.1. Another previous work was concerned about designing microservices architecture and finding the optimal size of these microservices, as shown in Section 2.5.2. Moreover, Section 2.5.4 describes a previous approach that was introducedto automate the process of transforming applications to use cloud-based services.

2.5.1 Benchmarking Heterogeneous Cloud Functions

The focus of this previous work was on evaluating the performance of cloud functions taking into account the heterogeneity of these function [MFGZ18]. In order to evaluate the performance, a cloud function benchmarking framework was developed. This framework consisted of two suites. The first suite was newly and specifically designed for this research and is based on Serverless Framework.The second suite uses the existing workflow engine HyperFlow [Bal16] which was extended to support cloud functions [Mal16] in order to be able to execute parallel workflow benchmarks. The described framework was applied to all the main cloud functions providers which are: AWS Lambda, Google Cloud Functions, Azure Functions, and IBM OpenWhisk. Finally, the results of the evaluation were discussed and made available online.
Each of the two suites was used for a specific purpose. The first benchmarking suite, that is based on Serverless Framework, was used to periodically execute the heterogeneous cloud function

benchmarks and to collect performance results of these executed benchmarks over a long period of time.Since the Serverless Framework provides a uniform way of setting up the deployment of cloud functions and supports the main providers, it was used to deploy the first benchmark suite. In order to deal with the heterogeneity of cloud functions, dedicated wrappers for native binarythat was executed by the function were created. The collected evaluation results were sent to and stored in InfluxDB time series database.
The second benchmarking suite, that is based on HyperFlow, was usedfor running parallel benchmarking experiments.HyperFlow is a lightweight workflow engine that can orchestrate complex large-scale scientific workflows including direct acyclic graphs [MFGZ18]. The executed cloud function, which is running on the cloud provider side, is a wrapper written in Javascript. This wrapper is responsible for running the benchmark and measuring the required time. Finally, the wrapper sends the results to the cloud storage.
In order to perform the experiment, each of the two benchmark suites was configured with a specific type of CPU-intensive benchmark. More precisely, the Serverless benchmark suite was configured with type of CPU-intensive benchmark that is focused on integer performance, and the Hyperflow suite was configured with a type that is focused on floating-point performance.The detailed experiment setup and explanation about how it was performed is provided in the original paper [MFGZ18].
During the first part of the experiment, which is the integer performance evaluation,the execution time t_b of the benchmark and the total request processing time t_r were measured. The results of this experiment illustrated as histograms,such that each histogram showed the performance evaluation of each cloud function provider.The histogram of the AWS Lambda shows that it is somewhat consistent and the CPU allocation is proportional to the function size, i.e. the memory. On the other hand, Google cloud functions does not compel strictly the limits of the performance, and it calls functions with smaller size using the faster resources in opportunistic way. Moreover, the performance in the IBM Bluemix does not depend on the function size and it has quite narrow distribution. On the other hand, the average execution times in Azure are slower and itsperformance has wider distribution.
Furthermore, through measuring the execution time t_b and the request processing time t_r, an estimate of the total overhead $t_0 = t_r - t_b$ can be obtained. The network latency to the AWS and Google clouds were not be able to be measured because these cloud provide access to functions via CDN infrastructure. On the other hand, the average round trip latencies to OpenWhisk and Azure were measured and the values were 117 MS and 155 MS respectively.

Regarding the second part of the experiment, which is the floating-point performance evaluation,it was applied only for AWS and Google cloud functions (GCF). The results were illustrated as a scatter-plots where circles densities represent the number of data points, such that AWS data consists of over 12,000 points, and GCF of over 2,600 points [MFGZ18]. Moreover, histograms of subsets of these data were drawn and inspected. On one hand, the histogram of AWS showed that the maximum performance increases with the function size in a linear manner. On the other hand, for GCF the performance points are clustered, i.e., for some tasks the performance increase linearly with memory, while other tasks achieve the highest performance independently of the function size.

In conclusion, the most important oversight that was obtained from this experiment is that both AWS and GCF shows that CPU allocation for cloud functions is proportional to the function size, i.e., to the allocated memory. Moreover, through inspecting the scatter-plots and histograms of

AWS and GCF, it was observed that on one hand AWS has linear performance growth with memory size for both lower bound and upper bound of the plot, and on the other hand, regarding GCF, the lower bound of the plot grows linearly while the upper bound is fairly constant [MFGZ18].

2.5.2 GranMicro: A Black-Box Based Approach for Optimizing Microservices Based Applications

The main idea of microservices architecture is to isolate business layer from presentation layer and to divide the business layer to set of small services called microservices which are invoked from one main interface called the application gateway [MMHP18]. In order to transform an existing web application to the microservices architecture, the application code should be split into group of web services each contains an isolated and decoupled part of the application code. The main issue while splitting the application code into web services is deciding how big the service should be, this design issue is called service granularity. Currently, deciding the service granularity is done through an iterative process [MMHP18]. In order to come over this iterative process, *GranMicro* approach was proposed. The main purpose of *GranMicr* is to plan for the optimal granularity services level in the early development cycle, which saves time and eliminate nonfunctional requirements leaks.

GranMicro is a black box based approach that aid software architects to decide the granularity of any new microservices evolved from transforming an existing web application to the microservices architecture. The main idea of *GranMicro* approach is to use web usage mining techniques to decide the most feasible granularity level. More precisely, *GranMicro* takes as an input the web access logs, applies mining techniques, extracts workload pattern, and finally builds the decision of granularity level and decomposing based on the extracted pattern.Some previous work started to research and investigate the fact that workloads have an impact over microservices based applications [UNO16]. On the other hand, the idea of using mining techniques in order to find the best granularity level is new and not had been researched before this work [MMHP18]. The mining process of web usage starts by extracting web access logs and preparing them for sessionizing, such that this sessionizing phase should be based on time stampsin order to extract *workload distribution pattern*. Then,*GranMicro* performs a clustering and weighting processin order to cluster these time periods into peaks and decide which web pages have high workloads. The specific clustering and weighting process performed by *GranMicro* is explained in details in the original paper [MMHP18]. Finally, the identified web pages with the highest workloads will have the highest priority to be split into microservices.

GranMicro approach was evaluated through applying it to a browsing scenario in Micro-Bookshop. More precisely, this Micro-Bookshop web application was transformed into two versions of webservices, one version through applying *GranMicro*, and the other version without applying this approach. The response time and performance of these two versions were compared. The result of the evaluation showed that both the response time and performance were improved in the version where *GranMicro* was applied. Moreover, the greatest improvement in the response time was noticed for web pages with the highest workload. In conclusion, *GranMicro* approach was considered as a good approach that helps the software architects to transform an existing web application to the microservices architecture.

2.5.3 Service Cutter: A Systematic Approach to Service Decomposition

The main purpose of this paper is to decompose systems to appropriate microservices using a systematic approach. As decomposing systems into discrete, accessible and autonomous microservices is a big challenge for software architects [GKGZ16], since the decomposition process is difficult and needs a deep understanding of the system architecture. Furthermore, it needs people with appropriate knowledge.

This paper proposed the first systematic approach that analyzes the code and applying a clustering algorithm in order to nominate decomposed and autonomous services. More precise, it benefits from the previous works in this domain and categorizes the coupling criteria. Afterward, it integrates graph clustering algorithm and analysis algorithms to form a systematic structure framework which is able to find microservices boundaries and nominate those microservices as a suggestion for decomposing the system.

2.5.4 Cloud Refactoring: Automated Transitioning to Cloud-Based Services

Transitioning a software application to use cloud-based services is not an easy process to be performed by hand [KT14]. This process requires the client to move the functionality to the cloud to be accessed as a remote cloud-based service, modify the code on the client side accordingly to invoke this service remotely, and finally he needs to detect and handle failures raised due to changing the code. This was as a motivation to develop a set of refactoring techniques which are automated transformations that substitute the manual process. These refactoring techniques, which are called *Cloud Refactoring*, had been implemented and added to Eclipse IDE. The introduced approach is not completely automatic, the only part performed by the programmer is to decide whether to transform the source code or not, but the actual transformations are performed automatically.This approach consists of two main parts: *The Recommendation Engine* and *The Refactoring Engine*.

Starting with the *Recommendation Engine*, it is used to decide which classes can be transformed to cloud-based services. It is consisted of two recommendation mechanisms which are *profiling-* and *clustering-based recommenders*. The *profiling-based recommender* performs a static program analysis and a runtime monitoring in order tocompute the coupling metrics of classes, choosing the classes that are least tightly coupled, and then suggesting them as parts of the application which can be transformed to cloud-based services.This recommender sorts the classes depending on their frequencies and execution duration, which enable the programmer to know which classes are computation-intensive and how frequently they are accessed [KT14]. On the other hand, the *clustering-based recommender* clusters all classes with similar functionalities, so that class clusters, whose functionality can be converted into cloud-based services, will be identified.

Regarding the Refactoring Engine, it is consisted of two techniques: (1) *Extract Service*, which automates the transformations required to convert the classes into remote service, and(2) *Adapt Service Interface*. Each functionality of cloud-based services is exposed though a public interface. Services providing equivalent functionalities are likely to have different interfaces [KT14], so one solution is to adapt one service's client interface bindings to be used for another service interface. The *Adapt Service Interface* automates this adaption. The *Cloud Refactoring* approach was evaluated through applying its techniques to transform three applications to use cloud-based services: two monolithic Java applications, and one commercial application by General Electric (GE). The results showed that Cloud Refactoring can be a robust tool for automatic transitioning of applications to use cloud-based services. Moreover, this powerful approach can decrease maintenance costs and increase the productivity.

The *Cloud Refactoring* technique has many advantages. One of the most important advantages is that through automating the transformations, the correctness of the modified program is guaranteed. Another strength point of this approach is that using the *Recommendation Engine*, the programmer will be informed of parts of the program that can be migrated to the cloud. On the other hand, this approach has some limitations. For instance, transforming tightly coupled applications require deep architectural changes which are not supported by this *Cloud Refactoring* approach. Moreover, the fault handling strategies implemented by this approach do not cover all the possible failures that might occur, so in such cases the programmer will need to handle these failures by hand.

3 Concept and Design

In this chapter, we will present our concept and proposed framework to provide insights for programmers and software architects about the suitability for moving to the serverless FaaS functions based on structural analysis and approaching to best use cases.

3.1 Motivation

This section shows us some challenges that face software architects and programmers when they need to move systems to the cloud computing. Additionally, it presents the real demands that encourage us to introduce a framework helping in assessing systems functionality for being deployed to FaaS model in the sake of upgrading systems and aligning them with new business need.

We mentioned in the introduction chapter (Section 1.1) that there are high demands on moving systems or parts of systems to the cloud environment. Since the increasing number of people conducting whole business based on automated systems and the increasing number of generated data in this big data era, both require systems and software having the ability to adapt with those increments as well as a powerful computing ability in order to enhance the system functionality performance and improve the efficient utilization of resources.
For the sake of helping to confront the previous challenges, moving to cloud computing becomes the concerns of many kinds of researches as we mentioned in the related work (Section 2.5), starting from decomposing systems to autonomous services or microservices that having effective performance in availability and scalability, and automating the transition of current systems to a cloud computing environment which faces a lot of difficulties in term of refactoring the systems architectures, furthermore, it might make the process costly and error-prone. Additionally, some researches focus on evaluating functions and services within cloud environments in order to figure out their capacity and adaptability with changes.

Moreover, the service model FaaS assists to achieve efficient utilization of computing resources and increase the performance of the current systems by moving the adequated functions to FaaS environment. Choosing which system functions and operations logic is valuable to move to serverless FaaS environment, that can make a change in performance efficiency and cost, requires specialists who have enough knowledge and expertise in the FaaS domain. Therefore, the concept proposed in this chapter aims to bridge this gap by providing a supportive approach that assesses the current functions and checks the suitability for running in FaaS environment.

Our proposed concept provides a decision support method for assessing the suitability of function to be deployed into the FaaS environment. This method introduces a framework helping programmers and software architects in choosing the appropriate functions that can be deployed in FaaS model. In principle, this framework might assist in moving current systems to FaaS environment, since moving all functions might be costly and inefficient, we need such a framework that checks the suitability

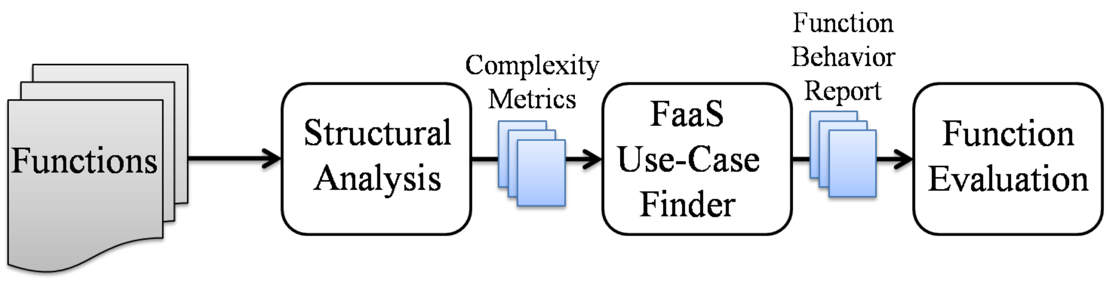


Figure 3.1: Main Process of our Concept

of function for running in FaaS environment. This framework consists of three main elements: a structural analysis model that analyzes the function source code and builds some important software metrics from it that could affect the function performance. Faas use-cases model that works on top of the structural analysis model to find the FaaS use-cases that work efficiently in FaaS environment and function evaluation model to process the result of previous models and link them with the required meta-data in order to eventually assess the function and give it a score clarifying the appropriability of function to be running in FaaS environment. The interaction between those three will guide us to answers for the research questions identified in the first chapter (Section 1.3) and carries an additional value for users and programmers. In the next sections, we will explain the framework architecture in details.

3.2 General Concept

In this section, we are going to present a general overview of the proposed solution. Firstly, we will explain how the proposed scenario starts.
According to what we mention in the previous section(Section 3.1), our concept comes to provide software architects some insights about the suggested functions whether they are appropriate for moving to FaaS. Figure 3.1 shows us a general process of our framework, where each step responds to an individual component in the framework architecture. Basically, the architecture of our framework consists of three main components that work together in order to assess the suitability of the function. Those components can be considered as the core engine of our method, and in the following, we define briefly the role of each component:

- ***Function Structural Analysis:*** a set of the analytical algorithms that are applied to function source code and operational logic in order to find some metrics measurement concerning code complexity, code performance, code suitability which have a direct impact to the performance of FaaS function. These metrics are used by the subsequent components in addition to other metrics that are imported from meta-data records as we'll see in detail later on.

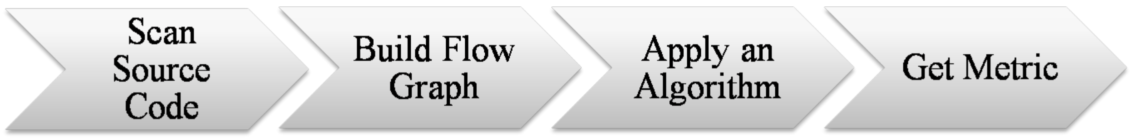


Figure 3.2: Process of Structural Analysis Model

- ***FaaS Use-Case Finder:*** the important component of the system which performs a specific analytics functionality on function source code and decomposes it into a set of FaaS use-cases depending on matching them with FaaS use-cases repository that contains efficient FaaS use-cases. Those matched use-cases play a major role in identifying how much the system function is applicable to move into and work on FaaS environment.
- ***Function Evaluation:*** responsible for gathering the required meta-data about the function behavior and linking them with metrics resulting from the preceding component models and applying a profiling feature. Eventually, it calculates a score that describes the overall suitability of function and operation logic to operate on the FaaS environment. This score can help the programmers and software architects on taking the right decision about moving an operational logic to the FaaS.

3.3 Static Structral Analysis Model

The soundness and integrity of the source code are critical for the efficiency of the code. This step of the process, presented in Figure 3.1, concerns in analyzing the structure of the function's source code. In this context, the analysis is considered as a static analysis which means examining the function code without actually executing the function. Basically, it scans the source code of the function and builds a matched flow graph if necessary in order to apply appropriate relevant algorithms that calculate some important metrics. Such algorithms might be cyclomatic complexity algorithm for calculating code complexity metric. Figure 3.2 shows the general process of structural analysis. The flow graph, which consists of nodes and edges, is built for the purpose of each algorithm. In general, the nodes might represent the instruction statements and the edges might represent the transfer of control between nodes. However, this representation might differ based on the applied algorithm.

Indeed, structural code analysis is an essential mechanism used by testing teams and software developers to keep application structure in shape and catch design flaw early, as well as to keep application architecture in a predictable and manageable state. Moreover, this technique provides an understanding of the code structure and can help to ensure that the code adheres to industry standards. More precisely, it can reveal errors that do not manifest themselves until a disaster occurs after software release. Nevertheless, we use this mechanism of analysis in our concept in order

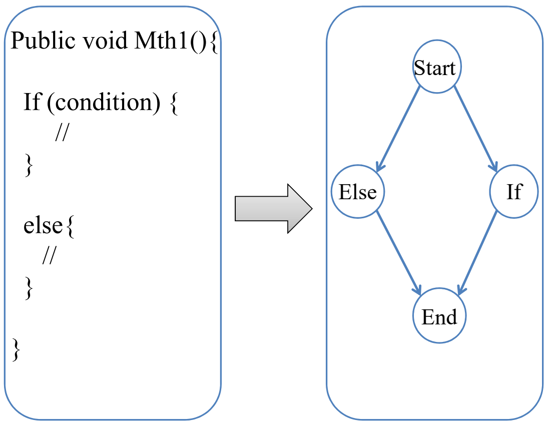


Figure 3.3: Example of Computing Cyclomatic Complexity

to calculate some software metrics that are considered critical factors by software architects in assessing the suitability of function for the FaaS environment. Therefore, our concept takes in its account some of these important metrics. In the following, we explain those considered metrics:

1. **Cyclomatic Complexity:** It is a software metric referring to the complexity of a program's source code. Precisely, it is the number of linearly independent paths through a program's source code, therefore, it is considered a quantitative measure [MMAS83]. In our context, it is the number of independent paths that exist between the input of the function and its output. Accordingly, it ensures that developers are sensitive to the fact that programs with high numbers of cyclomatic complexity are likely to be difficult to understand, as the higher the number the more complex the code.
   And in the sake of computing cyclomatic complexity, we need to build a control flow graph g(N,E) for the intended function, as N is the set of the graph nodes and E refers to the set of the graph edges. After scanning the code we apply the following algorithm to build the graph:

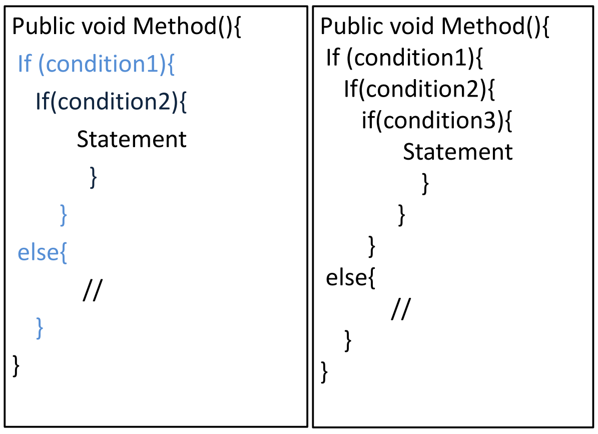
   - Create one node per code instruction.
   - Connect a node to each node which is potentially the next instruction.
   - Ignoring a recursive call or function call and consider it as normal sequential instruction. Otherwise, we do analysis for the function and its all called subroutines.

   Consequently, the formula of the cyclomatic complexity of a function based on its flow graph is simply:

   $$M = E - N + 2$$

   In Figure 3.3 we see an example of a control flow graph for a relevant function. Based on the graph, we compute the cyclomatic complexity as follow: M=4-4+2=2.

2. **Nested Block Depth:** Clearly this metric concerns in calculating how deeply nested the deepest statement in the intended function is. It measures the structural complexity of a method through the maximum nested block depth. Deep nesting shows a complicated design with too much control flow (in case of if/else nesting), computational complexity (in case of



```
Public void Method(){
 If (condition1){
    If(condition2){
           Statement
             }
        }
 else{
          //
    }
}
```

a

```
Public void Method(){
 If (condition1){
    If(condition2){
       if(condition3){
              Statement
                 }
             }
        }
 else{
          //
    }
}
```

b

Figure 3.4: Example of Nested Block Depth

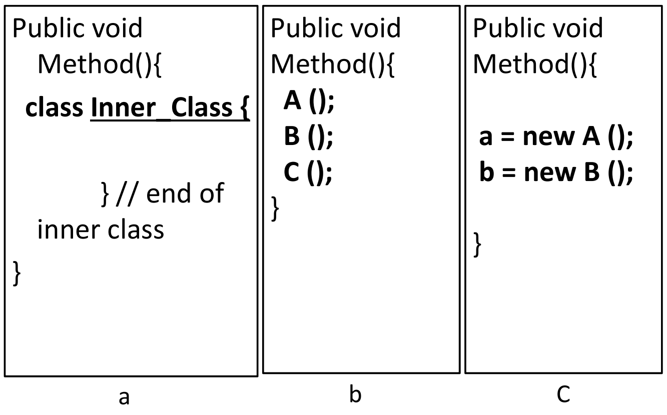
nesting for loops) or a combination of both. Figure 3.4 provides an example of two methods with different nested block depth, the figure (a) on the ride side contains a method with 2 nested blocks, whereas figure (b) contains a method with 3 nested blocks.

3. **Number of Inner Classes:** Most programming languages allows to define an inner class inside a function logic. Hence the function controls the life cycle of its inner class. But, this inner class might increase the complexity of the function, particularly, if the number of the inner classes exceeds a certain threshold. As this threshold could be depended on many factors like the power of computing resources and the memory capacity available for each function instance. The Figure 3.5 part (a) show us an example of defining an inner class inside a method.

4. **Number of Function Calls:** This metric refers to how many function invocations that the function body contains. When a function contains a lot of functions calls means the function is really dependent on those functions. Generally, each function call requires some additional work from the runtime environment which affects the complexity of the calling function. Furthermore, these invocations might increase the function executing time. Part (b) in Figure 3.5 shows us an example of function invocations inside the body of a function.

5. **Number of Referenced Classes:** We mean by referenced classes, the classes that are referred to in the body of the function. such references might increase also the complexity and affect the overall performance of the function. The less number of referenced classes the higher performance we get. Figure 3.5 part (c) presents an example of a method contains three referenced classes.

6. **Number of Parameters:** Eventhough, processing parameters doesn't have a big influence on the performance of the function because cloud computing provides very powerful computing resources. But the process of mapping the parameters by the API gateway of the FaaS environment might affect the performance if there are a large number of parameters.



```
Public void
   Method(){
 class Inner_Class {

          } // end of
   inner class
}
```

```
Public void
Method(){
 A ();
 B ();
 C ();
}
```

```
Public void
Method(){

 a = new A ();
 b = new B ();

}
```

Figure 3.5: Example of Inner Classes, Function Calls and Referenced Classes

7. **Number of Static Parameters:** Also this metric similar to the previous one. But a function with a large number of a static parameter might be not so efficient in a cloud environment.

Finally, we should pass these computed metrics to the FaaS use-case finder which is the next step of our process and continue the analysis process in order to achieve our goal in computing a score for a function that gives an indicator of how much it is suitable for the FaaS environment. Consequently, we want to mention that our concept is extendable, as simply we are able to add more metrics to our concept that might be useful for the purpose of the study.

3.4 FaaS Use-Case Finder

The FaaS use-cases finder model acts as the main engine in our proposed concept. Its primary task to find out FaaS uses-cases inside the function, those use-cases that are contributing to building up the operational logic of the function. Simply, FaaS use-case finder scans the function's source code and analyzes it in the purpose of finding FaaS use-cases. The FaaS use-cases in this context, mean efficient successful codes or type of operational logic that works in FaaS environment nearly perfectly. In general, those use-cases are common and similar between the FaaS vendors. Therefore, our concept generalizes those use-cases in an abstract manner and stores them in a FaaS repository. In Figure 3.7, we present a conceptual data model of those use-cases which will be explained in details in the following paragraphs.

Besides the main task of the FaaF use-case finder, it also provides some useful indicators to software architects regarding the orientation of the operation logic inside the analyzed function. As knowing such details, about the types of operations that compose the function as well as the percentage of them, might help software architects to enhance the function code and make it more appropriate for the purpose of the system goal, this enhancement could be by splitting the function

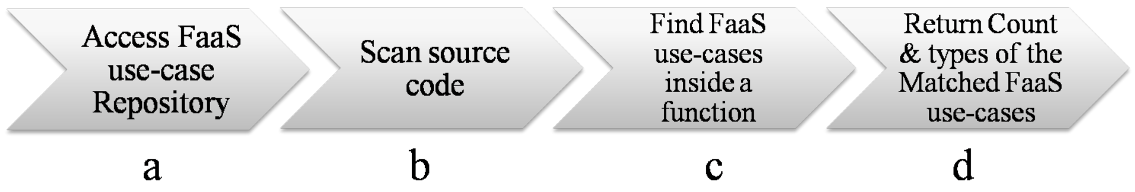

Access FaaS use-case Repository
Scan source code
Find FaaS use-cases inside a function
Return Count & types of the Matched FaaS use-cases
a
b
c
d

Figure 3.6: Process of FaaS Use-Cases Finder Model

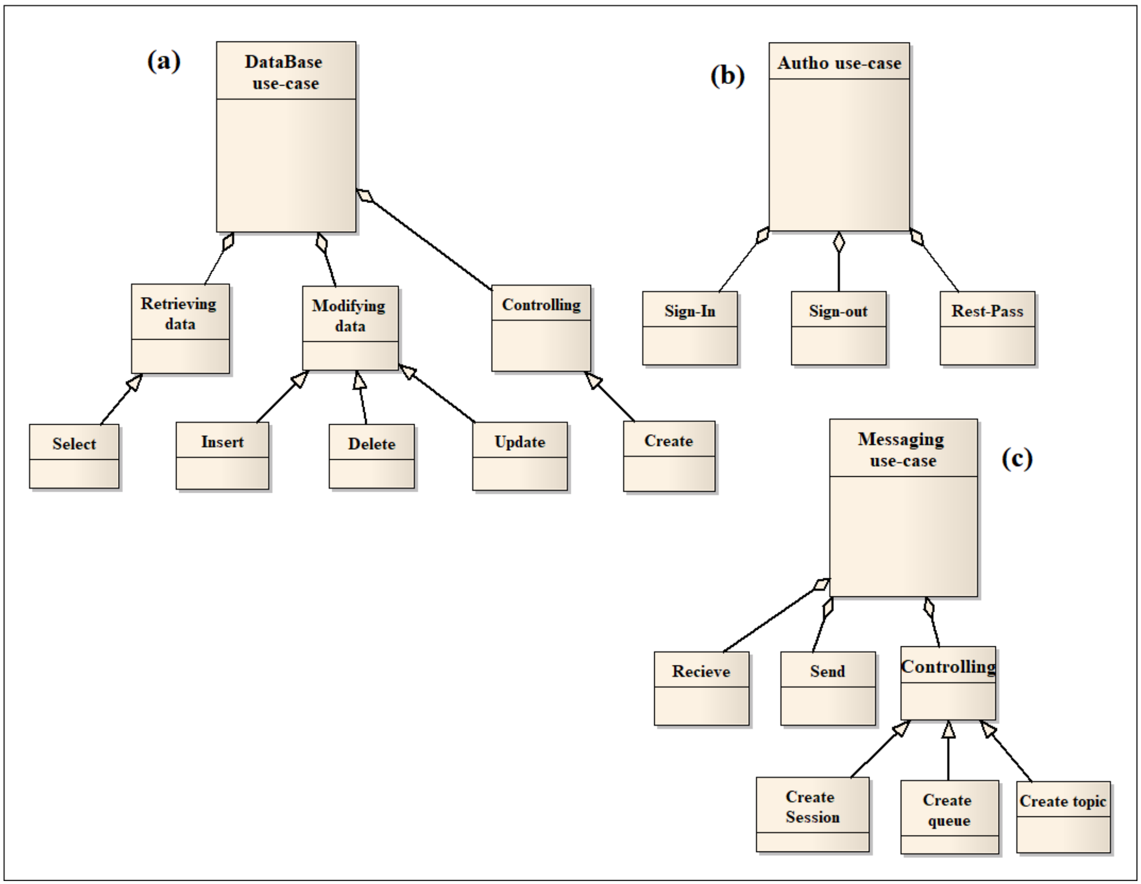


Figure 3.7: Conceptual Data Model of FaaS Use-Cases

or taking out parts of the code. However, this valuable information is provided by our concept for the further development process.

The FaaS use-case finder's working mechanism could be concluded into a group of steps that shows the expected sequence of activities. As listed in Figure 3.6, the FaaS use-case finder starts in part (a) by accessing the FaaS uses-cases repository and retrieve the relevant use-cases in order to know which cases he has to look for. Afterward, in part (b), the FaaS finder scans the function source code and build his own data structure that helps it to store the needed information. Figure 3.8

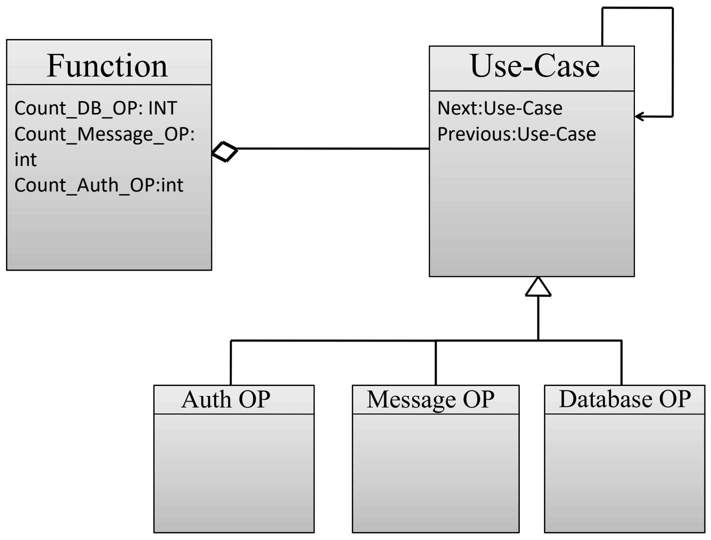


Figure 3.8: FaaS use-case finder Data Structure

shows us an abstract view of this data structure, where it mainly consists of a linked list from use-case type which could be a database operation, messaging operation or an authentication operation. This data structure is able to store all the needed information simply. On the other hand, it makes the navigation process through the data structure easier and faster. Additionally, it gives an idea about the sequence of those use-cases, which might be useful for understanding the code well.

In part(c) the process analyzes the information found in the previous step and matches it up with FaaS use-cases. This matching basically based on syntax matching approach that depends on keywords identification. In the final step in part (d), it provides the result of its analysis regarding the types of FasS use-cases that contributes to the function logic and number of those use-cases.
Generally, the FaaS use-cases are representing efficient operational logic, operations or methods' invokes that fits the FaaS environments and can invest the benefits of FaaS platform to improve their performance. Similarly to what we mentioned earlier, the existing FaaS vendors might vary slightly in defining the suitable use-cases but they still have the same common use-cases. Therefore, in the follwing we generalize those use-cases to three main types:

1. **DataBase Operations:** In fact, the database layer is popular and common between most of the systems and applications, essentially when we consider the legacy systems, as this layer allows the system to persistently store their data and information as well as analyzing this data and further operations. Based on the importance of the database layer, the systems regularly have to make many connections with such layer in order to push data-in, pull data out or control the data. Such connections might affect the performance of the systems, as from the technicgal side, the system is responsible to manage a connection, send the data, get the data and so on. Therefore, a system function that has many database operations might influence the overall performance, principally if the usage of function is high. Moving such a function

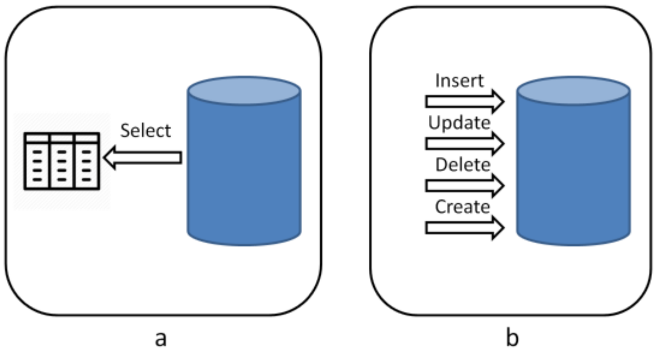


Figure 3.9: Database Operations

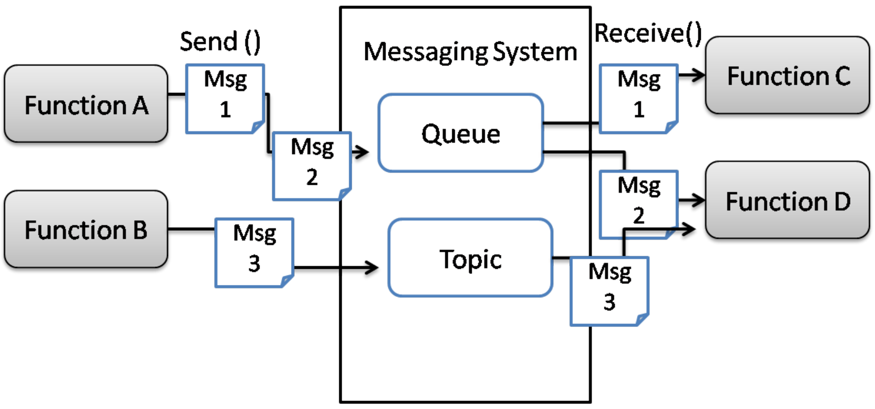


Figure 3.10: Messaging System Concept

to the FaaS environment might help to increase the performance even though there is a cost of transferring data to the cloud. However, It is a trade-off between the frequency of calling the function and the cost of transferring the data. Figure 3.9 shows us some of those database operations and their task in pushing in or pulling out the data from the database. By returning back to Figure 3.7 part (a), we present those operations in a conceptual model illustrating the relation between the operations and how they can be composed. Those operations might be:

- **Select**: Look up and retrieve all items whose characteristics fall into a given range.
- **Update**: Look up and upadte an item from the database
- **Delete**: Look up and remove an item from the database
- **Insert**: Add a new item to the database.
- **Create**: Add new component to database structure.

2. **Messaging System Operations:** With a highly increasing number of applications and systems

that work to achieve their purpose, the need for reliable communication and information exchange between those systems increased. Hence, the messaging system comes to provide a reliable layer that allows communications and messaging between variant systems and applications in a loosely coupled manner. As the messaging system approach allows the applications to exchange messages without the need to change their code or architecture, the only need is to plug-in onto the messaging system and start sending the messages. and the messaging system is responsible for storing and preserving the data until it is delivered to the consumer application. Figure 3.10 shows us an illustration of the general concept of the messaging system with two types of preserving the messages either by pish it down into a specific queue for one-to-one messaging or push it down to a specific topic for one-to-many communication. Moreover, a messaging system provides two methods of sending messages either in synchronous or asynchronous manner that based on the system needs. Therefore, such operations to open connection sessions, send messages and receive messages require additional processing in order to open, close and manage a connection stream to the underlying messaging system. Furthermore, FaaS function works based on an event-driven concept which it really fits this use-case, where the function can listen for a specific change in a message channel and once gets the message, it responds with specific actions. In the following, we' ll briefly enumerate some of the messaging system operations that can be used by inside a function operational logic. Additionally, Figure 3.7 part (c) shows the conceptual data model of this use case:

- create sessions, create a queue, create a topic: open a connection to the messaging system and create a new message channel.
- Send Message: push a message down to underlying message channel
- Receive Message: pull out the message from the message channel which might be in a synchronous or asynchronous manner.

3. **Authentication Operations:** Authentication is the process of determining whether someone is, in fact, who it declares itself to be. Authentication technology provides access control for systems by checking to see if a user's credentials match the credentials in the system database of authorized users or in a data authentication server. Users are usually identified with a user ID, and authentication is accomplished when the user provides a credential that matches with that user ID. Most users are most familiar with using a password that should be known only to the user. Indeed, authentication is a critical part of any system because companies or organizations can keep their systems secure by permitting only authenticated users (or processes) to access them. Once authenticated process succeeded, a user or process is usually subjected to an authorization process as well, to determine whether the authenticated user should be permitted access to a protected system.
   Generally, the operational logic of the authentication functionality is similar between different systems. When the same concept of authentication is applied, that would make some redundancy in re-implement such functionality. Moreover, security and reliability are very critical issues that should be taken into consideration once implement such functionality. Luckily, most of the cloud computing vendors supporting this functionality as FaaS function or BaaS service in order to make the authentication more secure and reliable (see **??**). Moreover, users can authenticate once to access multiple applications. Therefore, moving

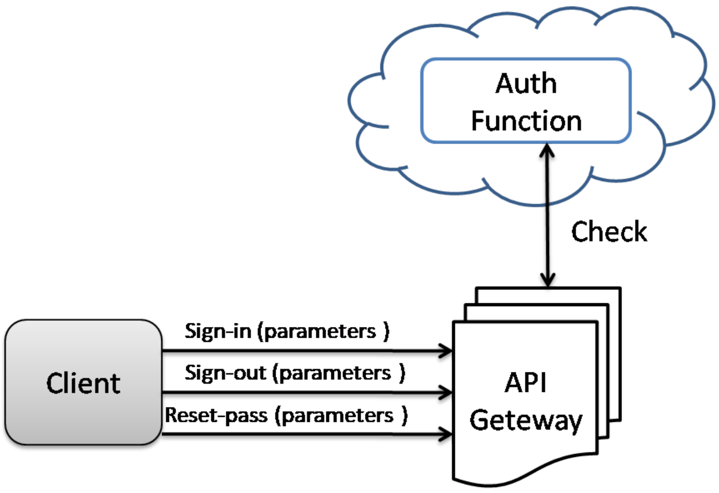


Figure 3.11: Authentication using FaaS

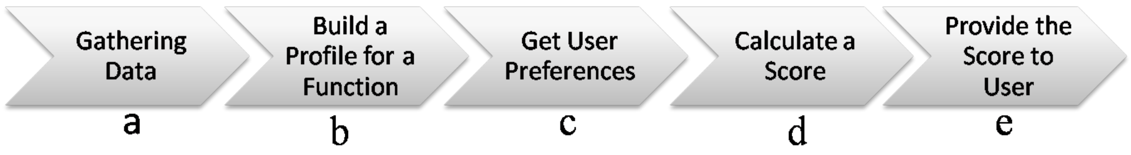


Figure 3.12: Process of Function Evaluation Model

this functionality to cloud computing and FaaS might help to reduce the code redundancy and hand over a part of user management to the cloud environment. Figure 3.11 presents the general concept of Authentication using FaaS, which illustrates that the client should invoke methods from gateway-API first, afterward the gateway-API launch the checking process using FaaS authentication function. The part (b) in Figure 3.7 shows the conceptual model for this use-case operations. And in the following, we mention some of the authentication operations that the client could use:

- Sign-in
- Sign-out
- Rest-Password

3.5 Function Evaluation Model

This is the last step in our concept process, where it has a fundamental task in the assessment process concerning the suitability of function to be deployed into the FaaS model. Basically, it adds a computation layer over the other layers which makes it fully depended on the data and results that come up from the previous models. It conducts processing the data and metrics concerning the

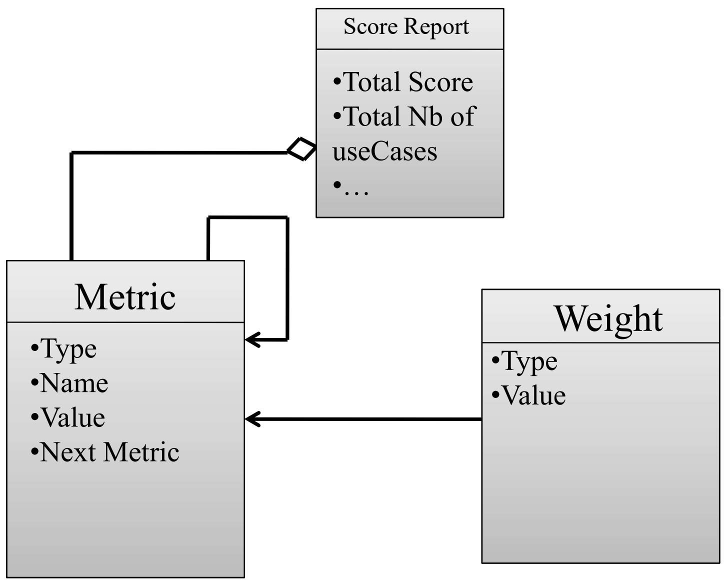


Figure 3.13: Evaluation Data Structure Model

intended function, more precisely, its essential task to combine the behavior of the function and its complexity and provides some insight to software architect about the analyzed function.
Figure 3.12 shows us a brief overview of the main steps of the function evaluation, it starts from gathering data phase in part (a) which addresses to bring in all the data and metrics resulting from the preceding steps. Actually, it fetches the complexity metrics from the structural analysis model, meta-data recorded from meta-data repository. Additionally, it uses the function behavior data that resulting from FaaS use-case finder. After gathering all the required data and metrics, the process moves to the next step, see part (b) in Figure 3.12, as it builds a simple and efficient data structure in order to conserve all the fetched data and prepare it for further manipulation steps. Figure 3.13 clarifying this data structure, where the score report data item stores the final report and aggregates other measurements. Indeed, this scoring item consists of multiple weighted metrics connected with each other in the form of a linked list in the sake of facilitating the score calculation process.

This data structure can be considered as a profile for each function. Usually, such a profile can be used for analyzing the overall behavior of the function in order to optimize it. Nevertheless, in our study, we will use it to implement an assessing approach on the top of it.
In step part (c) in our approach, we give flexibility for the users who are really involved deeply in upgrading systems, optimize system functions and have their own priority factors in assessing the suitability of a function for FaaS. So the users are able to contribute to this process by controlling some parameters and providing some weighting factors for the fetched metrics. This might make the assessing process fits the business needs and more realistic. As we can see in Figure 3.14, the calculating score activity takes the fetched list of metrics from the previous step as an input. Additionally, the user might define his own weighting factors for the fetched metrics based on

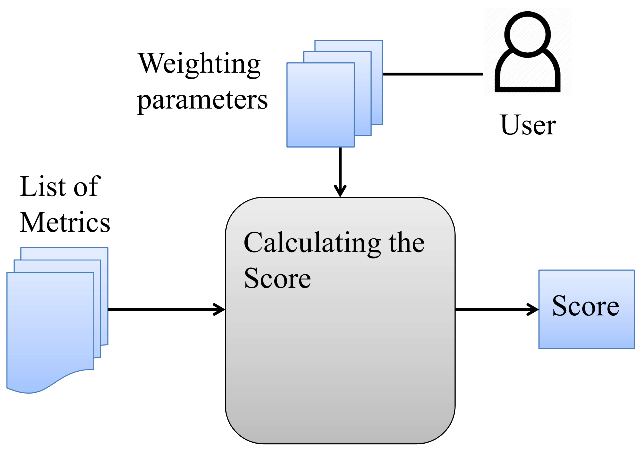


Figure 3.14: Score Calculation

his preferences and business need. Otherwise, the calculation activity uses the default weighting factors.

Afterward, the process starts calculating a score that presents the suitability of the analyzed function to be running in the FaaS environment by combining all the fetched metrics regarding function complexity and function behavior. In this step, we would apply a weighting algorithm for combining and weighting the gathered metrics in order to align it with business needs. In the following, we will discuss briefly how this weighting algorithm concept works. Let's assume that $(a_1, a_2, ...a_m)$ is a list of the fetched metrics, and $(w_1, w_2, ...w_m)$ is a list of relative weighting factors for our metrics where m is the total number of fetched metrics. Subsequently, the weighting algorithm combines both the metrics with the weighting factor with a view to calculate the score. In fact, there are many approaches for such combination and calculation, those approaches might differ in the way of addressing the weights factors. By example, the weighted sum model (WSM) provides its formula for calculating the score based on accumulating the weighted metrics as follow:

$$Score = \sum_{i=1}^{m} a_i * w_i$$

In the other hand, the weighted product model (WPM) subjects the score calculation based on multiplying the weighted metrics:

$$Score = \prod_{i=1}^{m} a_i^{w_i}$$

Nevertheless, in the next chapter, we will discuss the difference between those methods and what is the method that fits better with our solution.
Eventually, this calculated score from the previous steps can be provided to software architects. Basically, it is considered a recommendation that helps users to figure out the appropriates functions that are worth to move them to the FaaS platform, Moreover, it provides an insight into the attitude of the analyzed function.

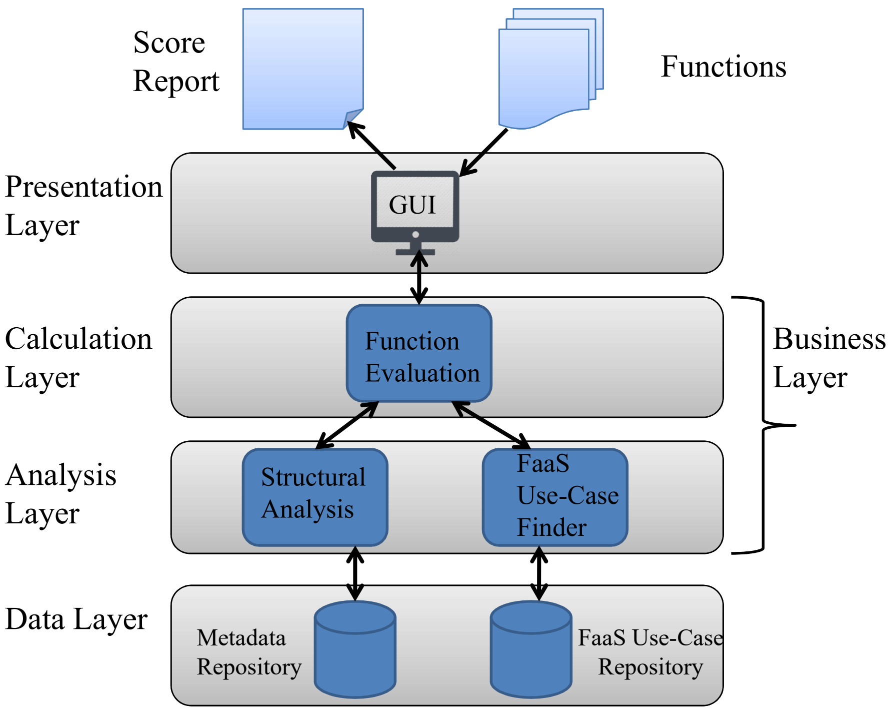


Figure 3.15: System Architecture Diagram

3.6 The System Architecture

In this section, we introduce a framework architecture supporting our concept in assessing function suitability for FaaS environment. This framework proposes a system consisting of multiple components. Thus, this section shows us how the interaction happens between this system components and what is the main role of each one.
Figure 3.15 presents an overview of the layered system architecture. Mainly, it consists of four layers like the following:

1. **Data Layer:** concerns to preserve the data and gathers it from different resources in order to prepare for further processing.
2. **Analysis Layer:** analyzes the complexity and the behavior of a function.
3. **Calculation Layer:** combines the analysis result and apply an algorithm in order to calculate a suitability score.
4. **Presentation Layer:** interacts with users. Basically, it takes from the user the intended functions as an input and finally provides users with a score report.

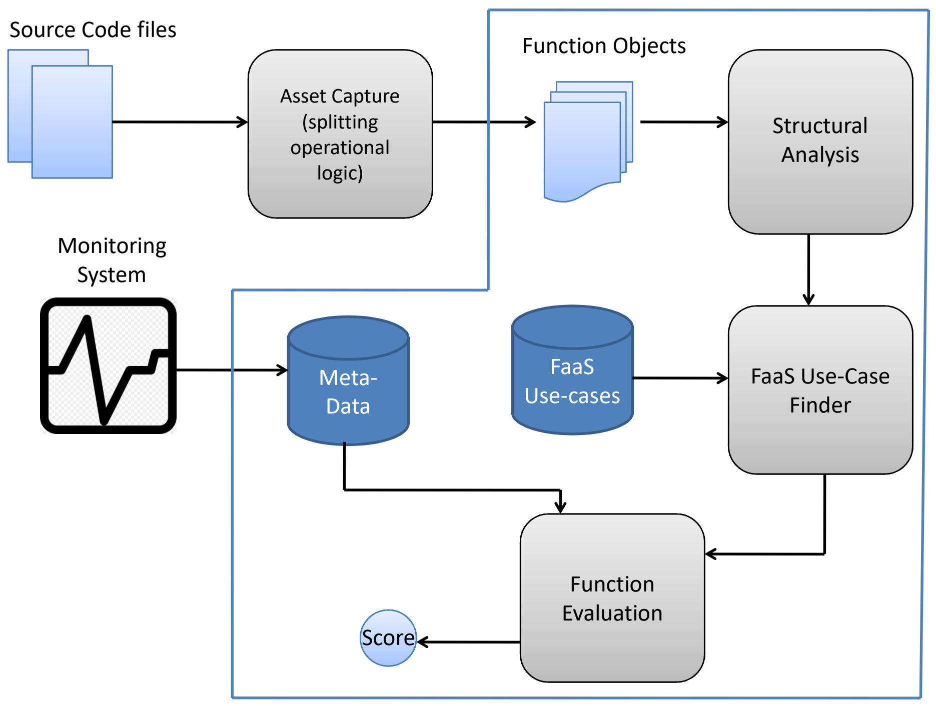


Figure 3.16: Flow Diagram

In order to understand our concept deeply, we will present the overall process of our concept and shows our main focus components in this process. Figure 3.16 shows the components of our concept starting from getting the source code files and ending by sending the recommendation score back to the user. The process passes through several steps, in the following, we will explain each step in details:

1. Source Code File: the source code files of a legacy or current system that programmers want to move parts of their functionality to cloud computing. precisely, they want to move into the FaaS environment. this source code files considered the starting point of the transferring process to the cloud environment.
2. Asset Capture: this component is an essential phase in the choosing FaaS functions process, whereas this component scans the source file code and applies an approach based on the asset capture concept that simply captures the assets inside the system and gathers all the event that concerns the assets in order to split the system functionality based on assets events [Fow04]. The result of this phase is a set of operational logic and functions that can work together to achieve the goal of their system. In other words, this phase split the functionality of the source system to a set of individual functions. Here, we want to mention that we choose this approach in taking out the individual functions from the legacy system because this approach considers events that might change the asset in its splitting process. In addition, FaaS functions depend on the events-driven approach for triggering. Absolutely, the user can change this component of code splitting to another approach that fits the target of the study.

3. Function Objects: a set of individual operational logic and functions resulting from the preceding component. In current new systems, these might represent the actual system source code functions but in a legacy system where functions might be mixed together and they don't have a clear separation between the system functionalities. As the function might contain a lot of operational logic that belong to different functionality, therefore the preceding component comes to separate them up to individual functions that support the target of the study. This phase is considered the starting point in building up our proof of concept as we assume that we have a pool of individual functions.

4. Monitoring System: we assume that we have a monitoring system in our concept that controls and observes the behavior of the intended system, the system we want to assess the suitability of its functions for FaaS. The monitoring system tracks the system functions in the runtime environment and helps understanding trends in usage or behavior and records some valuable measurements that are considered critical to take in our account in order to assess the suitability of function for FaaS. These measurements might include the number of function invocations, average executing time of the function, the time needed to start-up the function.

5. Metadata Repository: it stores the measurement values that come from the monitoring system component. It cleans, filters the data and makes it available for further analysis processes. This repository could be SQL database or JSON file or any other technology. In our concept we will take in our account these following measurements:

   - *Number of Function Invoke* (*NFI*): for each function we record the average number of function invoke per minute during the running phase. Such invoke might be internally from other system functions or from outside the system. This KPI is essential to be considered as an important factor if we want to transfer the function to the FaaS environment. accordingly, if the function is rarely invoked, it might be worthless to move it to the FaaS environment.

   - *Average Executing Time* (*AET*): also for each function, we record the average execution time that is needed to successfully execute the function. Although this time may differ when we move the function to the cloud environment, it still provides an indicator of how much the function execution is fast which also should be taking into consideration when we assess the functions.

   - *Start-up Time*(*ST*): the time needed for each function to be ready for executing starting from the time of receiving the function call. this might include initializing the variable and open the required reading streams if needed and so on.

6. FaaS Use-Case Repository: this repository stores efficient FaaS use cases that fit the FaaS environment, these use-cases might be database operations, messaging system operations, authentications operations. Where those use cases might help software architects in deciding about moving functionality to FaaS environment. In Section 3.4, we will discussed these use-cases in detail. This repository also might be a database or JSON file with a specific format or any other technology.

This architecture shows that our system is extendable, where adding a new component to the system, that might increase the assessing process accuracy, will not lead to change the whole architecture. moreover, we are able to extend our measurement data by considering new measurements that suit

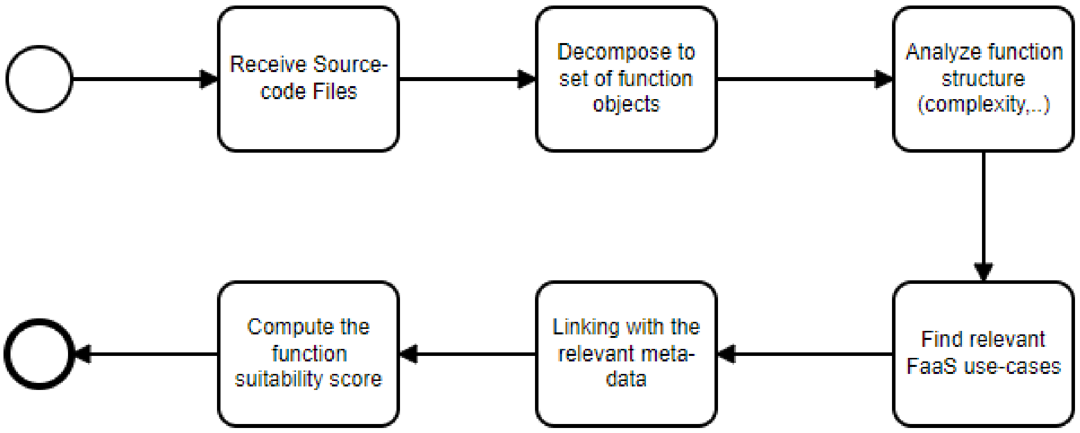

Receive Source-code Files
Decompose to set of function objects
Analyze function structure (complexity,..)
Find relevant FaaS use-cases
Linking with the relevant meta-data
Compute the function suitability score

Figure 3.17: process of assessing the suitability of function for FaaS

the purpose of the study.

In Figure 3.17 we illustrate the overall process of assessing the suitability of function to be deployed into FaaS model. The process starts from receiving the source code files of legacy systems or existing non-cloud systems, then if needed especially in legacy systems we decompose them to a set of functions. Afterward, we analyze the structure of the function and find out the complexity of it. The next step comes to analyze the function in term of finding a set of FaaS use-cases inside the function. Then, we are linking the previous result with the meta-data coming from a monitoring system to enrich the result. Eventually, the evaluation step comes to compute the overall score of how much suitable the function to be deployed in the FaaS environment. And here the user might control some parameters of the evaluation based on his priorities. We provide this score to the programmers or system architects in order to help them take the right decision about moving the function to the FaaS environment.

Actually, in this study, we are not able to study deeply all the steps because of the time constraints. Therefore we assume that we've already a pool of functions that need to be checked for their suitability regarding the running into FaaS environment, so our study addresses the process from the structural analysis step.

4 Implementation

In the following sections, we will explain a prototype implementation of the proposed approach in the previous chapter. As part of the thesis work, it is necessary to validate the proposed solution by an implementation.

In Section 4.1, we present the details about the prototype's architecture. Followed by Section 4.2 which briefly describes the class diagram and the benefits of this class structure are indicated. Moreover, Section 4.3 describes the used libraries, which provide the implementation of some steps of our framework. Followed by Section 4.4 which briefly describes the data layer that is implemented.

4.1 Architecture of the Prototype

The prototype implementation is built using the Java environment and the Mongo database. Figure 4.1 presents the architecture of the prototype which is based on the framework's architecture discussed in Section 3.6.

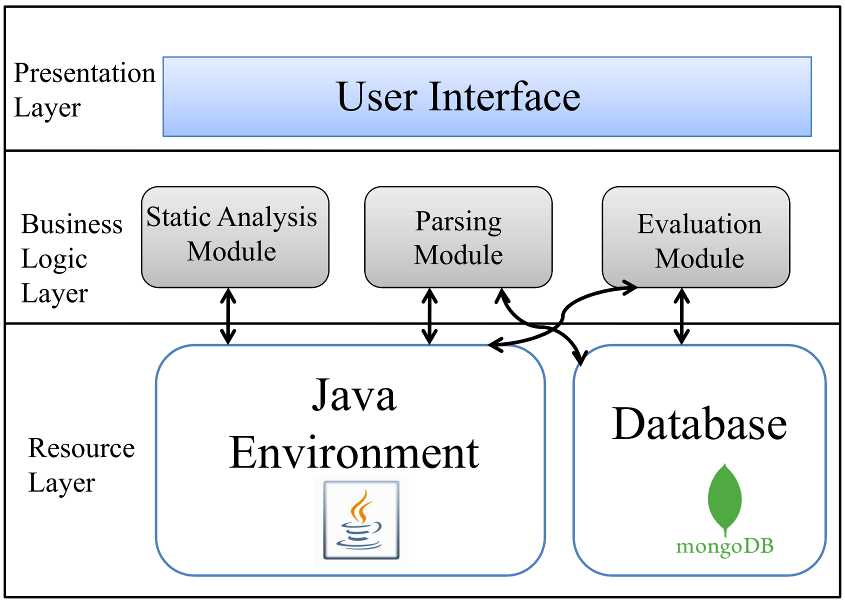


Figure 4.1: The architecture of the framework's prototypical implementation

We use java environment as the underlying infrastructure of our framework, therefore we use the Java programming language to implement our framework components that presented in Section 3.6, hence, our core components which represent the business layer of our concept can be divided to three modules. Those three modules work on the top of the java environment layer and the data layer, and corporate together in the sake of achieving the purpose of the framework by calculating an assessing score for the analyzed function. The Parse module which is responsible for scanning and parsing the function source code and calculates some code structure metrics. Whereas, the static analysis module comes to analysis the code and calculates code complexity metrics. Finally, the Evaluation Module gathers some calculated metrics as well as the metadata from the data layer and calculate the final score using a weighting algorithm. The following section will illustrate the functionality of each module in detail.

Regarding the data layer, we choose the MongoDB a non-relational document-oriented database. MongoDB stores data in JSON files which letting storing process simple and flexible for changes with offering expressive querying capabilities. Furthermore, choosing JSON files for storing the required data gave us a big advantage to easily build the structure of FaaS use-case data and change it based on the requirements. Moreover, storing metadata about the function behavior in schema-less document gave us more flexibility regarding changes in the structure of this metadata and simplifying the extension process, where adding new data and use-case don't force any changes in the previous data structure. Here, we should mention that most of our data operation is to retrieve data from the data repository where adding this data is not a part of our framework.

Finally, the user interface comes to be responsible for interacting with the user, it simply takes a function as an input from the user and the weighting factor preferences for some calculated code metrics. Those weighting factors that affect the assessing process of the function suitability for FaaS. Eventually, it returns the final score that represents the suitability of the analyzed function to be deployed into the FaaS environment.

4.2 Class Diagram

The implementation of our framework is divided into four main packages based on their major functionality. More specifically, the structural analysis and Faas use-case finder are implemented in packages called *parsing package* and *static complexity analysis package*. Furthermore, the evaluation step is implemented in the separate package called *evaluation package*. The final four packages, which is called *FaaS.suitabilty*, is the main package which imports all the other packages in order to control the workflow of the framework. Figure 4.2 illustrates the package diagram as described above. Each package contains the corresponding Java interfaces and classes which provide the functionalities of the respective package. The subsequent subsections introduce the class diagram of each of these four packages. The main package FaaS.suitabilty is described in the final Section 4.2.4, and the benefits of using this package design with the corresponding classes designs are indicated.

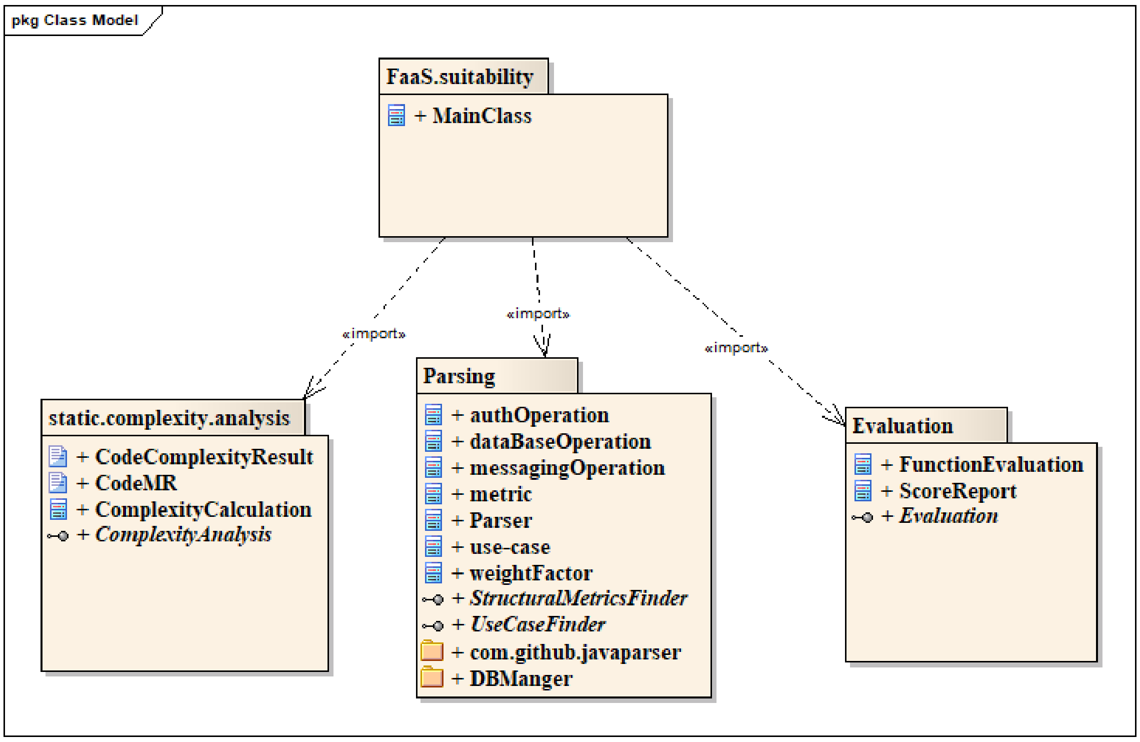


Figure 4.2: Package Diagram

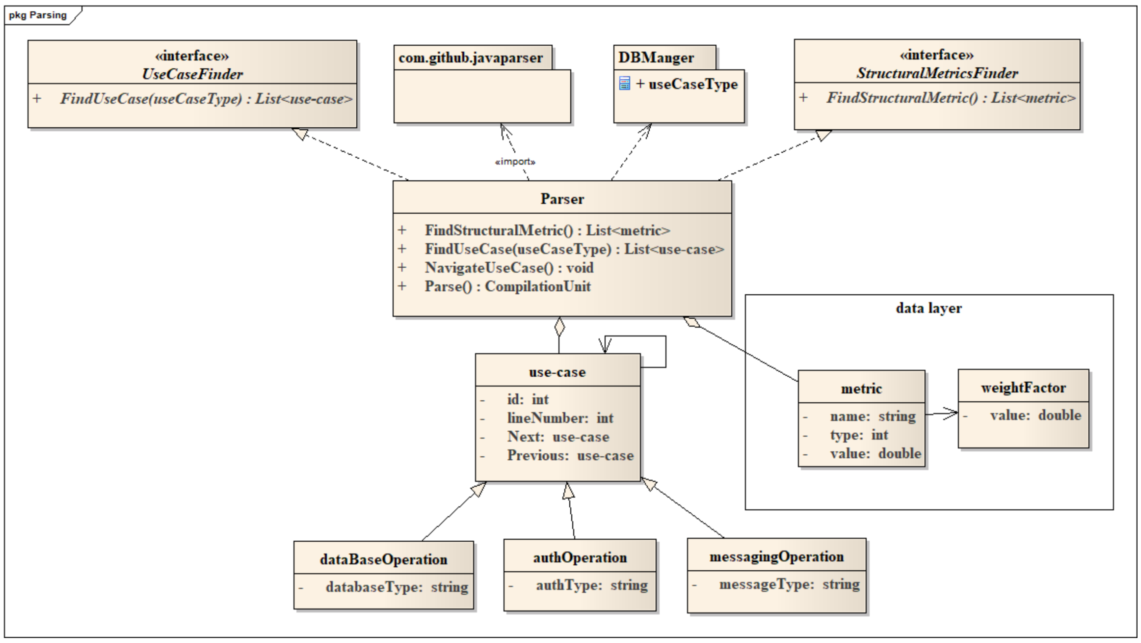


Figure 4.3: Parsing Package Diagram

4.2.1 Parsing Package

The package Parsing, shown in Figure 4.3, contains the implementation of parsing the function and finding some structure code metrics as well as finding the FaaS use-cases inside the parsed function. More precisely, this package contains two main interfaces which provide the main functions needed for performing the parsing and analysis. The first Interface called *structuralMtericFinder* which provides the main function need for performing structural code analysis and find some metrics. This function is called *findStructuralMetrics()* that returns a list of metrics concerns the structure code of the function. The second main interface called *UseCaseFinder* which also introduces the main function needed for finding FaaS use-cases inside the analyzed function. This function is called *FindUseCase(useCaseType)* which return a list of founded use-case that matches the type useCaseType as we will see below.

The engine of this package is the *Parser* class. Basically, this class depends on *com.github.javaparser* package which provides Java library facilitating parsing the java code and navigates through its structure. And we will see the principle of such a library in the Section 4.3. More specifically, the function *Parse()* provided by this class is fully depended on *parse()* function from *com.github.javaparser library*, this function that parses java code and returns a tree data structure called *CompilationUnit* representing the parsed code including all statements and expressions, as we will illustrate this mechanism in Section 4.3. Moreover, the parser class implements the above-described interfaces. On one hand, the implemented function *findStructuralMetrics()* navigates the tree data structure *CompilationUnit* that is resulted from *parse()* function, and calculates some structural metrics through this navigation, by example, it calculates the number of inner classes and number of referenced classes, as described in Section 3.3. The principle of this calculation is to check the type of each visited tree node as illustrated in Algorithm 4.1.

Algorithm 4.1 Finding Structural Metrics

```
function FINDSTRUCTURALMETRICS(CompilationUnit Tree)
    for Each node N from Tree do
        if N is an Inner Class then
            NumberOfInnerClass++
        else if N depends on Referenced Class then
            NumberOfReferencedClass++
        else CheckOtherCases
        end if
    end for
    List<metric> ← BuildCodeMetrics(NumberOfInnerClass,NumberOfReferencedClass,...)
end function
```

Afterward, this function stores the calculated metrics in a list of the specific data structure that represents the metrics and their value as subjected in Figure 4.3.
On the other hand, the *Parser* class re-implements the function *FindUseCase*, this function that is responsible about searching inside function code for a specific use case type. More specifically, this function takes a specific use-case type as an input and looks for all the use cases that are declared inside the function code and have this specific type. By example, it might search for all database use cases that are parts of the function logic. The principle used for searching about such use cases

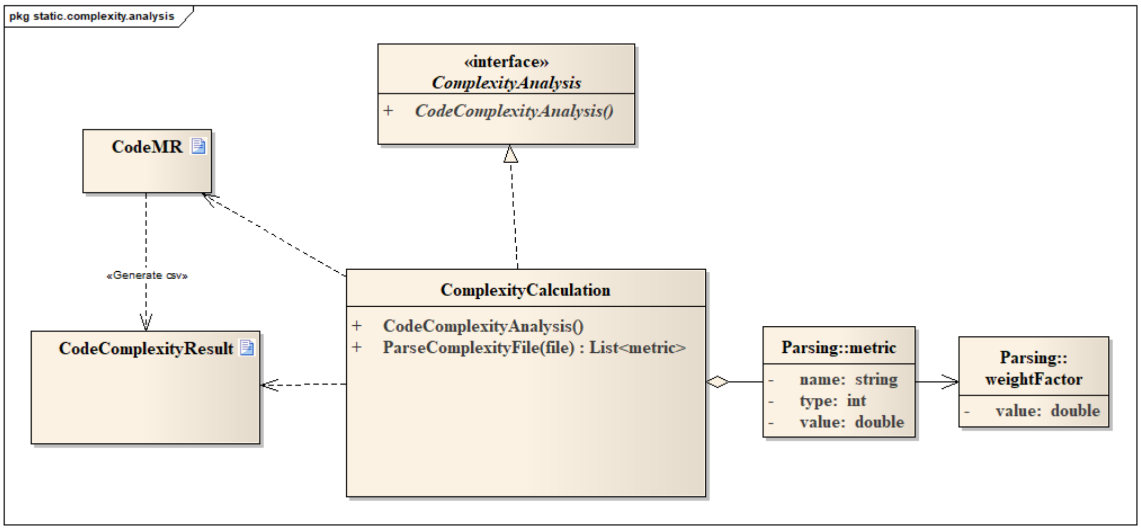


Figure 4.4: Complexity Analysis Package Diagram

is similar to what we mentioned in the previous function. Where it also depends on navigating through the code tree, but it uses the *visit()* function which provided by the *com.github.javaparser* library. As this function facilitates scanning the code tree and finds all the matched node that we are looking for. More details about *visit()* function provided in section Section 4.3.
The result of *FindUseCase* function is a linked list from founded FaaS use-cases, where the data structure of such use-case simplifies the navigation through the list and figuring out all the metric we are looking for. By example, aggregating all use cases of selecting from a database. Figure 4.3 illustrates the data structure of those FaaS use-cases.
Moreover, the *Parser* class uses the package *DBManger* which is responsible for all database operations with the underlying data layer. In this context, it retrieves all the predefined FaaS use case types from the data repository and provides them to the parser. Those use-case types in our case concern the database operations, messaging system operations, and authentication operations. More details regarding the structure of the use-case type will be illustrated in Section 4.4.

4.2.2 Static Complexity Analysis Package

The *static.complexity.analysis* package, as shown in Figure 4.4, contains one interface called *ComplexityAnalysis* which provides the main function needed for code complexity analysis. This function called *CodeComplexityAnalysis()* and it provides the actual functionality.
Furthermore, this package contains the main class called *ComplexityCalculation*. This class implements the interface *ComplexityAnalysis*, such that it provides the functionality of calculating code complexity metrics. The computation of code complexity is done by calling the corresponding functionality from a tool called codeMR. This tool responsible for computing code complexity metrics and generating a CSV file containing the resulted metrics. Such metrics like cyclomatic complexity and nested block depth. The functionality in this tool calculates the responding graph from the analyzed code, afterward, it calculates the complexity metrics based on that graph as we described it in Section 3.3. Hence, we choose this tool because it facilitates generating the corresponding code graph and fetching the code complexity based on it. Eventhough, we are able

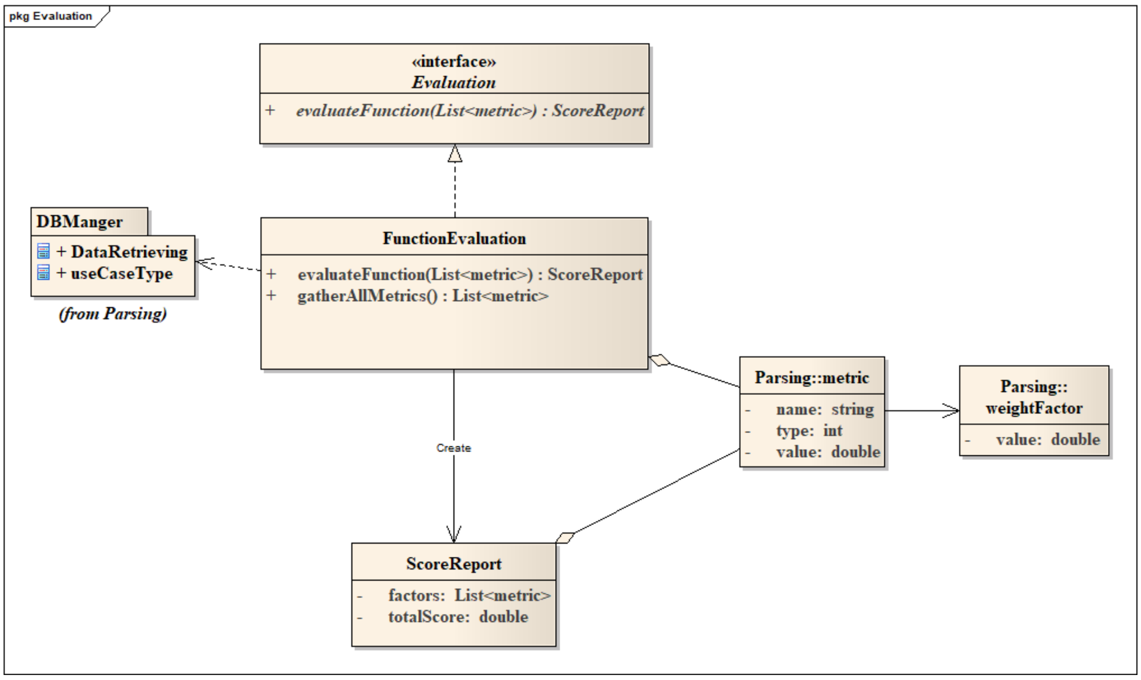


Figure 4.5: Evaluation Package Diagram

to change this tool by reimplementing its algorithms.
Moreover, the function *ParseComplexityFile()* in the class *ComplexityCalculation* simply parses the CSV file that is generated from the codeMR tool and builds a list of complexity metrics as shown in Figure 4.4.

4.2.3 Evaluation Package

The *Evaluation* package, as shown in Figure 4.5, contains one interface called *Evaluation* which provides the main function needed to evaluate the analyzed function and calculate a score represents the function suitability for FaaS environment. This function called *evaluateFunction()* and it provides the actual functionality. Furthermore, this package contains one main class called *FunctionEvaluation*. This class has a function called *gatherAllMetrics()* that collects all the previous calculated metrics: 1- the code metrics from structural analysis; 2- the code complexity metrics; 3- and retrieves metrics from metadata repository regarding the average number of invocation and execution time, those metrics coming from monitoring system as we described in Section 3.6. So this function uses some functionality from *DBManger* package in order to retrieve the metadata from the JSON file and build a list of respective metrics.
Moreover, The class *FunctionEvaluation* implements the above-described interface and is responsible for performing the calculating of the score using one of the weighting methods. The function *evaluateFunction()* takes the list of all the calculated and collected metrics and navigates through it in the sake of calculating the final score that represents the suitability of the function for FaaS environment. The calculation depends on a weighting method that we mentioned the principle of it in Section 3.5. And here we select the weighted product model (WPM) as an approach for calculating the score since the weighted sum model (WSM) is not fits our case, because the weighted

sum model (WSM) considers all the weighted factors from the same unit where it is not suitable for multidimensional problem. On the other hand, the WPM approach has a structure that eliminates any units of measure. And furthermore it is more efficient for comparing two lists of factors [Tri00]. Therefore the function evaluateFunction calculates the score as shown in Algorithm 4.2.

Algorithm 4.2 Evaluate Function for FaaS Model

```
function EVALUATEFUNCTION(List<metric> M)
    Score ← 1
    for Each metric from M do
        Score ← Score* $(metric- > Value)^{metric->weightFactor->value}$
    end for
    Score ← normalize(Score)
    ScoreReport ← BuildScoreReport(Score,List<mtetic>,...)
end function
```

The resulted Score should be normalized before we can consider and return it to the user. Afterward, the function builds an object from the *scoreReport* class in order to preserve the result with all the metrics contributing to the result.

4.2.4 FaaS.suitabilty Package

The FaaS Suitability package, as shown in Figure 4.2, consists of only one runnable class called MainClass. This class controls the whole workflow of all steps of our framework. The main function performs the following steps: First, it enables the user to enter the function that needs to be checked for its suitability in the FaaS environment. Second, it calls the function readWeightingFactors() that reads the weighting factors that are defined by the user to reflect its preferences in assessing the function. Third, it continues by calling the functions corresponding to applying each step of our concept.
More specifically, the main function continues by calling the following functions: (1) the function FindStructuralMetric() contained in the package Parsing, which performs the first step in our concept, (2) the function FindUseCase() also contained in the package Parsing, in order to find all the FaaS use cases inside the function logic, (3) the function CodeComplexityAnalysis() contained in the package static.complexity.analysis, which computes the code complexity metrics for the analyzed function, and finally (4) the function evaluateFunction() contained in the Evaluation package, which is responsible for calculating the final score that represents the function suitability for FaaS environment.
Obviously, the design that is shown in the package diagram in Figure 4.2 and described in detail in the previous subsections, has many benefits and advantages. One of the main advantages of this design is its maintainability. More clearly, in case one of the steps performed by our framework has to be changed or modified, this leads to only local changes in the corresponding class contained in the respective package without affecting any other package. Furthermore, the simplicity of the design is another important advantage. This simplicity comes from the fact that the implementation of each step of our concept is encapsulated in an interface, i.e., each interface is responsible for performing only one step in our concept. Another important advantage is the scalability and extensibility of the design. More precisely, using other complexity analysis approaches only requires adding a new

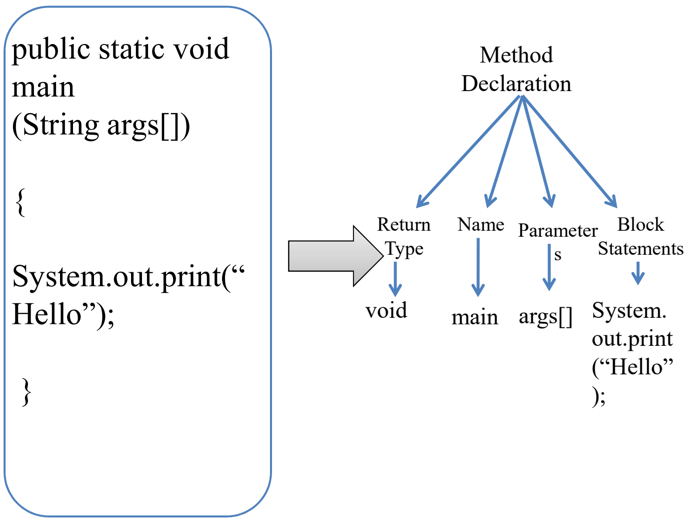

public static void main (String args[])
{
System.out.print("Hello");
}
Method Declaration
Return Type
Name
Parameters
Block Statements
void
main
args[]
System.out.print("Hello");

Figure 4.6: Example of Parse Function

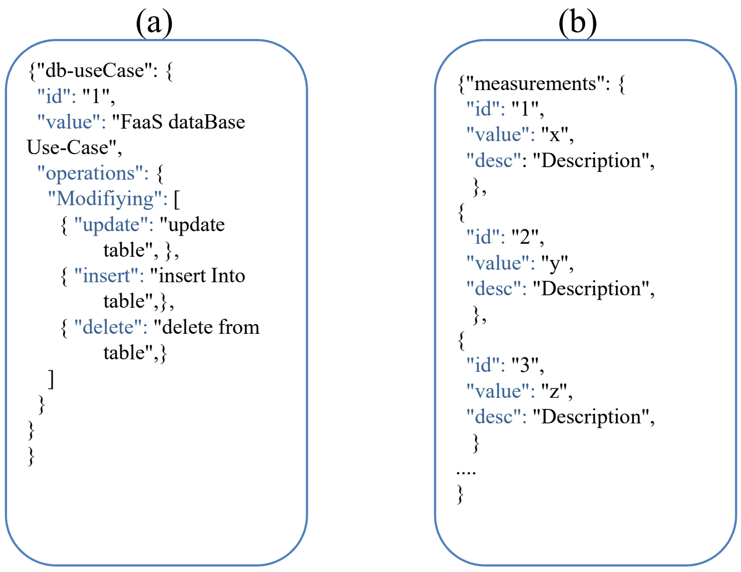
implementation for the ComplexityAnalysis interface, i.e., adding only a new class in the package static.complexity.analysis such that this class implements the respective interface. Additionally, these modifications require changing only the class initiation in the MainClass, i.e., require changing only one line of code. For instance, adding new class providing a new implementation for the Evaluation requires initiating in MainClass an object from this newly added class instead of the object from the old class. The addition of a new class is done easily due to the simplicity of the whole design.

4.3 Libraries and Tools

Our framework uses some existed libraries as a basis for building the following processing steps. More specifically, in order to perform the parsing step, it is necessary to implement or to use implementation for parsing java code. Therefore, we use the package *javaparser* to facilitates the code parsing process. Furthermore, we used the tool *codeMR* that conducts the code complexity analysis. In the following, we will briefly illustrate the role of each of them.

4.3.1 Javaparser Library

This library provides the main functionality that allows to parse a java code and convert it to java objects which can be represented in a Java environment. More specifically, it converts the java code to an Abstract Syntax Tree (AST). Moreover, it provides the ability to manipulate the underlying structure of the source code and facilitates navigation through this abstract syntax tree where it provides a powerful data structure for such a tree. Basically, this structure consists of nodes and



(a)

```
{"db-useCase": {
 "id": "1",
 "value": "FaaS dataBase
Use-Case",
 "operations": {
  "Modifiying": [
   { "update": "update
        table", },
   { "insert": "insert Into
        table",},
   { "delete": "delete from
        table",}
  ]
 }
}
}
```

(b)

```
{"measurements": {
 "id": "1",
 "value": "x",
 "desc": "Description",
  },
{
 "id": "2",
 "value": "y",
 "desc": "Description",
  },
{
 "id": "3",
 "value": "z",
 "desc": "Description",
  }
....
}
```

Figure 4.7: Example of Storing The Data

each node might have its own properties and a list of children that can be accessed by childrenNodes reference. The root of the tree doesn't have any parent whereas the leaf node doesn't have any children. Figure 4.6 shows us an example of a code tree that represents the respective code.

Additionally, this library provides two main functions, the *Parse()* function that scans the java code, generates the respective code tree and returns back this tree. Whereas, the *visit()* function allows the user to define patterns that he is looking for through the tree and then the function returns all the founded matched patterns to the user. This function facilitates the searching and moving through the tree and allow the user to focus only on his own functions.

4.3.2 CodeMR Tool

CodeMR is a software quality and static code analysis tool that helps software developer to produce better code and better quality products. It checks code complexity, cohesion, and coupling while investigating code quality. This tool can be plugged into the coding editor. Once it is invoked, it checks the code syntax and analyzes it in order to provide a list of metrics that might help the developer to understand the code and enhance it [cod].
Indeed, we use this tool in our concept for facilitating the calculation process concerning the code complexity i.e. cyclomatic complexity and nested block depth. This tool generates a graph from the respective code using the algorithm mentioned in Section 3.3. Afterward, it calculates the metrics concerning code quality. These resulted metrics can be exported in many formats like a CSV file or HTML. In our prototype, we used the CSV file for exporting the resulted metrics.

4.4 Data Layer

In this section, we will illustrate the conceptual data model used by our prototype for achieving its purpose in assessing functions. More precisely, in our prototype, we have two main types of data we process them. The data that represents the FaaS use-cases and the data that is generated from the monitoring system as we described in Section 3.6.
In Figure 3.7, we present the conceptual data model for the FaaS use-cases. This conceptual model is serialized using JSON files and stored in MongoDB. As we can see, this model is extendable that allows us to add more use-cases simply without affecting the previous use-cases or obliging us to change the structure of data. Figure 4.7 part (a) shows us a simple example of a JSON file contains a FaaS use-case.

Regarding the data that is generated from a monitoring system, we assume that it is stored using a JSON file as well. Simply, this file contains a list of measurements that are observed from the monitoring system during the runtime, and Figure 4.7 part (b) presents an example concerning the structure of these metadata. This structure keeps our prototype adaptable for any change that might occur in the data structure.

5 Conclusion

Moving to a cloud computing environment is a very essential topic in the big data era, where a huge amount of data needs to be processed every day. In addition, the number of systems users is increasing sharply. Those main challenges encourage business owners to move current systems and legacy systems to a cloud computing environment. Many reference process models, such as the cloud refactoring approach [KT14], exist in order to help users in moving the applications to a cloud environment. Choosing a suitable cloud service model is an essential step in the process of applications transition to a cloud environment in order to optimize the performance and reduce the overheads. Function-as-Service (FaaS) is one of cloud service models that appeals users due to its important role in managing, initializing and scaling the functions. In addition, it provides an attractive pricing model that considers only the actual usage of the underlying infrastructure. Even though FaaS looks appealing for users due to its advantages, but choosing the functions that really fit the FaaS environment is not trivial. In fact, not all functions might be suitable for FaaS model and are able to make a change in performance and cost.

In this thesis, we introduce a new concept that aims to help software architects and developers to assess the suitability of function for running in FaaS environment. This concept is considered as a decision support system that assists users to check the fitness of moving a function into the FaaS environment and provides a score representing the suitability of a function for being deployed to FaaS model. In Chapter 2 we discuss the background concepts and research topics which are relevant for this research. Afterward, in Chapter 3, we introduce our framework for assessing a function by checking its syntax structure and behavior. Starting from taking a function, that needs to be checked, as an input for our framework and applying a static structural analysis on this function to calculate some important software metrics concerning the complexity and structure of the function that might influence the suitability of this analyzed function for being run in FaaS model. Next, we check the behavior of the function by checking what types of FaaS use cases are defined inside its operational logic. Furthermore, we take in our consideration the metadata that is observed during the function runtime. Eventually, our framework combines all the previous factors and calculates a score representing the suitability of function for running in FaaS environment. This score can be provided to users in order to support their decision regarding moving the function to a cloud environment. Lastly, in Chapter 4, we describe the details of the framework's prototypical implementation.
Within this thesis, the function semantic analysis concerns have been discarded. This analysis might have an influence on the function assessing process and the result. Moreover, the security aspect should be considered in the assessment process to make the result more realistic and valuable, since security issues are some of the biggest concerns in cloud computing. Therefore, future work can focus on those two aspects and continue the research in order to fully automate the function selection and transition process.

Bibliography

[ama] amazon.com. *aws lambda pricing*. URL: https://aws.amazon.com/lambda/pricing/ (cit. on p. 30).

[Bal16] B. Balis. “HyperFlow: A model of computation, programming approach and enactment engine for complex distributed workflows”. In: *Future Generation Computer Systems* 55 (2016), pp. 147–162. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2015.08.015. URL: http://www.sciencedirect.com/science/article/pii/S0167739X15002770 (cit. on p. 32).

[BJJ10] S. Bhardwaj, L. Jain, S. Jain. “Cloud computing: a study of Infrastructure As A Service (IAAS)”. In: *International Journal of Engineering and Information Technology* 2 (Jan. 2010), pp. 60–63 (cit. on pp. 18, 20).

[cod] codemr.co. *CodeMR Software Quality Tool*. URL: https://www.codemr.co.uk/ (cit. on p. 63).

[DTM10] W. Dawoud, I. Takouna, C. Meinel. “Infrastructure as a service security: Challenges and solutions”. In: *2010 The 7th International Conference on Informatics and Systems (INFOS)*. Mar. 2010, pp. 1–8 (cit. on p. 18).

[EIST17] E. Eyk, A. Iosup, S. Seif, M. Thömmes. “The SPEC cloud group’s research vision on FaaS and serverless architectures”. In: Dec. 2017, pp. 1–4. DOI: 10.1145/3154847.3154848 (cit. on pp. 23–25).

[Eiv17] A. Eivy. “Be Wary of the Economics of SServerless"Cloud Computing”. In: *IEEE Cloud Computing* 4.2 (Mar. 2017), pp. 6–12. ISSN: 2325-6095. DOI: 10.1109/MCC.2017.32 (cit. on p. 31).

[Fow04] M. Fowler. *Asset Capture*. 2004. URL: https://martinfowler.com/bliki/AssetCapture.html (cit. on p. 51).

[GKGZ16] M. Gysel, L. Kölbener, W. Giersche, O. Zimmermann. “Service Cutter: A Systematic Approach to Service Decomposition”. In: *Service-Oriented and Cloud Computing*. Ed. by M. Aiello, E. B. Johnsen, S. Dustdar, I. Georgievski. Cham: Springer International Publishing, 2016, pp. 185–200. ISBN: 978-3-319-44482-6 (cit. on p. 35).

[HBKH09] J. Hurwitz, R. Bloor, M. Kaufman, F. Halper. *Cloud Computing For Dummies*. For Dummies, 2009. ISBN: 0470484705, 9780470484708 (cit. on pp. 18, 20, 22).

[IBM] IBM. *IaaS, PaaS and SaaS – IBM Cloud service models*. URL: https://www.ibm.com/cloud/learn/iaas-paas-saas (cit. on pp. 17, 20).

[Ibr] A. Ibrahim. *IaaS Cloud service models*. URL: https://www.slideshare.net/AsmaaIbrahim2/fundamental-concepts-and-models (cit. on p. 19).

[Joh17] M. R. John Chapin. *What Is Serverless?* O’Reilly Media, Inc., 2017. ISBN: 9781491984178. URL: https://www.oreilly.com/library/view/what-is-serverless/9781491984178/ (cit. on pp. 26–29).

[KT14] Y.-W. Kwon, E. Tilevich. “Cloud refactoring: automated transitioning to cloud-based services”. In: *Automated Software Engineering* 21.3 (Sept. 2014), pp. 345–372. ISSN: 1573-7535. DOI: 10.1007/s10515-013-0136-9. URL: https://doi.org/10.1007/s10515-013-0136-9 (cit. on pp. 11, 12, 35, 65).

[Mal16] M. Malawski. “Towards Serverless Execution of Scientific Workflows – HyperFlow Case Study”. In: Nov. 2016 (cit. on p. 32).

[mar] marutitech.com. *SERVERLESS ARCHITECTURE THE FUTURE OF BUSINESS COMPUTING*. URL: https://www.marutitech.com/serverless-architecture-business-computing/ (cit. on p. 29).

[MFGZ18] M. Malawski, K. Figiela, A. Gajek, A. Zima. “Benchmarking Heterogeneous Cloud Functions”. In: *Euro-Par 2017: Parallel Processing Workshops*. Ed. by D. B. Heras, L. Bougé, G. Mencagli, E. Jeannot, R. Sakellariou, R. M. Badia, J. G. Barbosa, L. Ricci, S. L. Scott, S. Lankes, J. Weidendorfer. Cham: Springer International Publishing, 2018, pp. 415–426. ISBN: 978-3-319-75178-8 (cit. on pp. 32–34).

[MG+11] P. Mell, T. Grance, et al. “The NIST definition of cloud computing”. In: (2011) (cit. on p. 11).

[MMAS83] T. MacCabe, McCabe, Associates, I. C. Society. *Structured testing*. Tutorial Texts Series. IEEE Computer Society Press, 1983. URL: https://books.google.de/books?id=vtNWAAAAMAAJ (cit. on p. 40).

[MMHP18] O. Mustafa, J. Marx Gómez, M. Hamed, H. Pargmann. “GranMicro: A Black-Box Based Approach for Optimizing Microservices Based Applications”. In: *From Science to Society*. Ed. by B. Otjacques, P. Hitzelberger, S. Naumann, V. Wohlgemuth. Cham: Springer International Publishing, 2018, pp. 283–294. ISBN: 978-3-319-65687-8 (cit. on p. 34).

[MS18] A. Møller, M. I. Schwartzbach. *Static Program Analysis*. Department of Computer Science, Aarhus University, http://cs.au.dk/˜amoeller/spa/. Oct. 2018 (cit. on p. 32).

[ore] oreilly.com. *PaaS Cloud service models*. URL: https://www.oreilly.com/library/view/cloud-analytics-with/9781788839686/ad932aae-3c5f-46fb-a784-89992156afd1.xhtml (cit. on p. 21).

[pac] packtpub.com. *SaaS Cloud service models*. URL: https://hub.packtpub.com/cloud-computing-services-iaas-paas-saas/ (cit. on p. 22).

[Rob18] M. Roberts. *Serverless Architectures*. 2018. URL: https://martinfowler.com/articles/serverless.html (cit. on pp. 27, 29).

[SFL11] S. Stuckenberg, E. Fielt, T. Loser. “The impact of software-as-a-service on business models of leading software vendors : experiences from three exploratory case studies”. In: *Pacific Asia Conference on Information Systems (PACIS) 2011*. Ed. by P. Seddon, S. Gregor. Queensland University of Technology, Brisbane, Qld: Queensland University of Technology, July 2011. URL: https://eprints.qut.edu.au/43815/ (cit. on p. 21).

[SZG+08] W. Sun, X. Zhang, C. J. Guo, P. Sun, H. Su. “Software as a Service: Configuration and Customization Perspectives”. In: *2008 IEEE Congress on Services Part II (services-2 2008)*. Sept. 2008, pp. 18–25. DOI: 10.1109/SERVICES-2.2008.29 (cit. on pp. 21, 22).

[Tri00] E. Triantaphyllou. “Multi-Criteria Decision Making Methods”. In: *Multi-criteria Decision Making Methods: A Comparative Study*. Boston, MA: Springer US, 2000, pp. 5–21. ISBN: 978-1-4757-3157-6. DOI: 10.1007/978-1-4757-3157-6_2. URL: https://doi.org/10.1007/978-1-4757-3157-6_2 (cit. on p. 61).

[UNO16] T. Ueda, T. Nakaike, M. Ohara. “Workload characterization for microservices”. In: *2016 IEEE International Symposium on Workload Characterization (IISWC)*. Sept. 2016, pp. 1–10. DOI: 10.1109/IISWC.2016.7581269 (cit. on p. 34).

[Win] M. Winters. *CNCF Serverless Whitepaper v1.0*. URL: https://github.com/cncf/wg-serverless/tree/master/whitepapers/serverless-overview (cit. on pp. 23–25).

[WLY+10] L. Wang, G. von Laszewski, A. Younge, X. He, M. Kunze, J. Tao, C. Fu. “Cloud Computing: a Perspective Study”. In: *New Generation Computing* 28.2 (Apr. 2010), pp. 137–146. ISSN: 1882-7055. DOI: 10.1007/s00354-008-0081-5. URL: https://doi.org/10.1007/s00354-008-0081-5 (cit. on p. 12).

[WVZX10] G. Wei, A. V. Vasilakos, Y. Zheng, N. Xiong. “A game-theoretic method of fair resource allocation for cloud computing services”. In: *The Journal of Supercomputing* 54.2 (Nov. 2010), pp. 252–269. ISSN: 1573-0484 (cit. on p. 17).

[XL08] M. Xin, N. Levina. “Software-as-a-Service Model: Elaborating Client-Side Adoption Factors (December 22, 2008)”. In: *Proceedings of the 29th International Conference on Information Systems, R. Boland, M. Limayem, B. Pentland, (eds), Paris, France, December 14-17, 2008* (Sept. 2008). DOI: https://ssrn.com/abstract=1319488 (cit. on pp. 21, 23).

All links were last followed on June 03, 2019.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature