applied sciences

Article

The FaaS-Based Cloud Agnostic Architecture of Medical Services—Polish Case Study

Dariusz R. Augustyn [1], Łukasz Wyciślik [1,*] and Mateusz Sojka [2]

[1] Department of Applied Informatics, Faculty of Automatic Control, Electronics and Computer Sciences, Silesian University of Technology, 44-100 Gliwice, Poland
[2] Independent Researcher, 43-100 Tychy, Poland
* Correspondence: lukasz.wycislik@polsl.pl

Abstract: In this paper, the authors, based on a case study of the Polish healthcare IT system being deployed to the cloud, show the possibilities for limiting the computing resources consumption of rarely used services. The architecture of today's developed application systems is often based on the architectural style of microservices, where individual groups of services are deployed independently of each other. This is also the case with the system under discussion. Most often, the nature of the workload of each group of services is different, which creates some challenges but also provides opportunities to make optimizations in the consumption of computing resources, thus lowering the environmental footprint and at the same time gaining measurable financial benefits. Unlike other scaling methods, such as those based on MDP and reinforcement learning in particular, which focus on system load prediction, in this paper, the authors propose a reactive approach in which any, even unpredictable, change in system load may result in a change (autoscaling) in the number of instances of computing processes so as to adapt the system to the current demand for computing resources as soon as possible. The authors' main motivation for undertaking the study is to observe the growing interest in implementing FaaS technology in systems deployed to production in many fields, but with relatively little adoption in the healthcare field. Thus, as part of the research conducted here, the authors propose a solution for infrequently used services enabling the so-called scale-to-zero feature using the FaaS model implemented by the Fission tool. This solution is at the same time compatible with the cloud-agnostic approach which in turn helps avoid so-called cloud computing vendor lock-in. Using the example of the system in question, quantitative experimental results showing the savings achieved are presented, proving the justification for this novel implementation in the field of healthcare IT systems.

Keywords: medical IT services; FaaS; Fission; cloud computing; Kubernetes; scale-to-zero

check for updates

Citation: Augustyn, D.R.; Wyciślik, Ł.; Sojka, M. The FaaS-Based Cloud Agnostic Architecture of Medical Services—Polish Case Study. *Appl. Sci.* **2022**, *12*, 7954. https://doi.org/10.3390/app12157954

Academic Editor: Eui-Nam Huh

Received: 6 July 2022
Accepted: 4 August 2022
Published: 8 August 2022

1. Introduction

The financial outlay incurred in the healthcare sector is increasing year by year [1]. At the same time, a growing share of information technology supporting the healthcare sector can be observed. IT, in a positive sense, is breaking into more and more areas of healthcare [2], such as, but not limited to:

- ERP;
- Treatment decision support systems;
- Diagnostic imaging;
- Telemedicine and telecare;
- Bioinformatics;
- PACS/RIS systems;
- ADT systems;
- Clinical data repositories.

Each of the above applications of computer technology requires computing power that is expensive and not without impact on the surrounding environment. Not surprisingly, more and more IT systems, including those supporting healthcare, are being located in specialized data processing centers called computing clouds, which due to their high specialization and large scale are able to lower the total costs of ownership and at the same time lower the environmental footprint compared with similar IT systems being deployed to local, scattered, proprietary server rooms owned by individual medical facilities. Reducing the cost of cloud computing is possible mainly due to the so-called economies of scale and optimization of computing to reduce the consumption of electricity which is the subject of active research [3]. However, just implementing a system in cloud computing does not yet guarantee the optimal use of the resources a given cloud provides. Therefore, it is important to first develop an architecture for such a system suitable for taking advantage of all the benefits that cloud computing can offer. The first step may be to ensure compliance with the concept of cloud-native, the use of which brings a number of advantages such as flexible deployment options, or less effort for the development and operation of a given system. Following cloud-native architecture can also provide longer-term benefits in the form of the ability to fine-tune a system deployment variant after it has already been initially deployed. An example here could be easily changing the class of the S3 object data store or moving some of the services implemented as server based on the FaaS (i.e., Function-as-a-Service) model. A good case study from the ground of the Polish healthcare sector of such an architecture system evolution could be the solution responsible for recording and storing medical records, which was presented in a former article [4]. The services of this solution are primarily responsible for the safe storage of metadata and relevant medical documents using PIK HL7 CDA level 3 standard (https://www.cez.gov.pl/HL7POL-1.3.1.2/plcda-1.3.1.2/plcda-html-1.3.1.2/plcda-html-1.3.1.2/index.html, accessed on 13 April 2022). The described system not only enables the creating and storing of medical documents, but due to the integration with the system P1 (https://www.cez.gov.pl/pl/main-page-en, accessed on 13 April 2022) (which indexes medical documents), it indirectly enables them to be found and downloaded. The nationwide P1 system plays a role based on the service locator architectural pattern for the CDR, i.e., it redirects registered medical entities to the appropriate CDR service. Following cloud-native architecture can also provide longer-term benefits in the form of the ability to fine-tune a system deployment variant after it has already been initially deployed. An example here could be changing the class of the S3 object data store or moving some of the services implemented as server-based to the FaaS (i.e., Function-as-a-Service) model.

The system of services described in 2021 provided functionalities in the field of handling medical records for small- and medium-sized clinics (hundreds) and hospitals (several). Currently (first half of 2022), the system is additionally used by several hospitals, where a hospital information system (HIS) is a client of services that handle either storing the medical documentation or records of medical events. Additionally, the functionality of handling questionnaires dedicated to the patients or staff of healthcare units is being introduced.

Several months of monitoring the services' ecosystem allows us to conclude that their load distribution over time varies significantly. For example, in the case of the service of storing and sharing documentation, round-the-clock traffic is generated, but with greater intensity during daily working hours. In turn, when it comes to the medical event handling service, it is mainly fed with data during the day, but the load of the entire system does not result from accepting requests but from the continuous batch processing of them (data validation and successful data registration in the remote, independent P1 system). Therefore, we can talk about a large, fairly constant load on the system handling a common queue of medical events. On the other hand, the survey service is characterized by an unpredictable load distribution. We may observe the so-called usage peaks but also significant and frequent periods of a lack of utilization.

For these three different types of services, it was, therefore, necessary to develop a technical model of operation and an appropriate technological stack that would ensure sufficient performance and, at the same time, would be economically optimal in terms of resource usage.

The average variability of the load (in the time scale of an hour or tens of minutes) of the medical documentation storing service allowed for the economically optimal control of the offered computational capacity by standard horizontal autoscaling mechanisms. The HPA (https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale, accessed on 13 April 2022) mechanism available as part of the Kubernetes infrastructure allows for the achievement of satisfactory results. The medical events service has a high, 24/7, very slow-varying workload (in the timescale of hours) that is fairly predictable in general. This allows the use of a slow-varying time-scheduling CRON-like scaling mechanism (https://varlogdiego.com/schedule-pods-to-start-and-stop-in-kubernetes-by-date-and-time, accessed on 13 April 2022). Optionally, it may be supported by the aforementioned HPA mechanism to handle possible temporary overloads.

However, the time distribution of the utilization of the survey service is very unpredictable. The load of the service is occasional, although some survey campaigns may appear, which can generate a demand for increased performance. Therefore, it would be useful to apply scaling mechanisms with a minimum number of instances of 0 (the so-called scale-to-zero mechanism). The use case scenario of this service fits perfectly with the FaaS concepts (i.e., scaling functions from zero to large values, with a low complexity of functions). There are lots of such elements in the system in question which are compliant with FaaS specificity, both when it comes to infrastructure (authentication and authorization) and domain parts (defining questionnaires and submitting questionnaires). As the services composing the described system are largely hosted by the Kubernetes platform (specifically, by GKE (https://cloud.google.com/kubernetes-engine, accessed on 13 April 2022)—a managed orchestrator of containers with microservices) provided by a cloud service provider (Google Cloud Platform), the functionalities to adapt to the FaaS processing model should be technically compatible and integrated with K8s.

To sum up the problem statement, in this paper, the authors try to propose the new system architecture for some occasionally used medical services to achieve a significant reduction in cloud infrastructure resource consumption in a way to stay cloud-agnostic. This could be fulfilled by a transition of services deployed as plain K8s pods to the FaaS model but still hosted by the same Kubernetes environment, which was selected as the execution environment for all medical cloud services of the system in question. Further, the article justifies the selection of the concrete FaaS environment/framework, describes the whole architecture of the solution, presents the assumptions for implementation (recommendations for the technology stack) and shows the experimental results of the efficiency of the proposed solution.

2. Related Work

For some time now, we have been witnessing an industrial revolution driven by the development of technology. The healthcare field is part of this industry and its development is not at all lagging behind. Based on successive paradigms, there is now talk of Industry 4.0 and 5.0 [5], and scientific research on the development of healthcare in these areas is confirmed by numerous publications [6].

Among the many developments in IT, technologies such as IoT, blockchain, AI and cloud computing have become key drivers in recent years. This is also happening in the healthcare field [7–9].

It has been less than eight years since AWS—one of the biggest players in cloud computing—publicly released the first FaaS product. Since then, the popularity of this model of conducting computations has gained many followers, found many practical applications and become the subject of research conducted by many researchers. Quite surprisingly, however, it seems that relatively few practical applications or research papers are related to the healthcare sector, even broadly defined.

Serverless computing applications in the healthcare sector can be roughly divided into several application areas. A strong group here is bioinformatics computing. For example, one of the biggest challenges in the field of omics computing is the data sequencing process itself, which is computationally very demanding. Executing such computations in a standard model requires the reservation of computing power and the self-management of virtual servers. However, research shows [10] that, using AWS Lambda, it is possible to conduct the all-against-all pairwise comparison among all unique human proteins in approximately 2 min, at a cost of less than USD 1, while performing the same task on a typical laptop computer would take more than 8 h. An interesting application of serverless computing is shown in paper [11], where building a high-quality annotation corpus (MedTator) for biomedical and clinical research applications is described. Due to involving a serverless approach, the authors develop the application with an intuitive and interactive user interface that focuses on the core steps related to corpus annotation, such as document annotation, corpus summarization, annotation export and annotation adjudication. In other studies [12], the authors identify bottlenecks that stand in the way of efficient omics data processing. In their view, it seems to become problematic to store data, preprocess them or integrate data from multiple sources. To streamline the computational process, they propose the use of the Amazon Serverless Lambda service and discuss the benefits but also the risks of doing so. More information on applications of the FaaS model for bioinformatics computing can be found in a dedicated literature review [13,14].

Another area of application in serverless computing, characterized by the possibility of batch data processing, is the analysis of medical data, in particular imaging diagnostics. Based on it, the authors of [15] present a serverless model for highly parallel file processing applications. They also describe a middleware implementation that supports the execution of customized execution environments based on Docker images on AWS Lambda. Their results prove that the combination of a high-level programming model with the scalable capabilities of AWS Lambda makes it easy for end-users to efficiently exploit serverless computing for the optimized and cost-effective execution of loosely coupled tasks. Other research [16] shows the concept of the serverless edge data analytics platform and application model and discuss its main design requirements and challenges, based on real-life healthcare use case scenarios. The authors hope that the proposed model will switch the current view of centralized premise and cloud real-time analytics into more distributed, edge, ubiquitous, real-time analytics, in which the data's value will not be lost at the edge and all computing layers will be used evenly.

Since the process of diagnosing patients is primarily data-driven, it is the issue of medical data storage that makes up a large share of the overall scientific interest in the health sector. For example, based on a case study of storage services to manage medical imagery, the authors of [17] describe research which results in the development of a storage mesh architecture to create and operate reliable, configurable and flexible serverless storage services for heterogeneous infrastructures. The topic of the distributed file system is also taken up by the authors of the article [18] which proposes a robust, available, scalable and serverless solution structure for storing large amounts of data in the medical field. Big data sets also pose challenges related to their efficient and effective searching. Here, too, researchers [19] refer to the concept of serverless computing. They develop the ScanMedicine module which is a new searching system dedicated to providing healthcare professionals, patients and researchers with easy access to the intelligence underpinning health technology innovations. Once again, the serverless part of the architecture assumes the use of AWS Lambda services.

Proposals for serverless solutions can also be found in research on telemedicine-like solutions. An example of this is a chatbot developed in India that helps people potentially suffering from COVID-19 to find a vacancy in the hospital. However, for its development, the authors of [20] do not use the classic FaaS approach, in which they could imperatively program logic, but use a number of services from the Microsoft cloud, such as Power Virtual

Agent or Power Automate. Another chatbot proposal [21], however, also created to limit the spread of COVID-19, uses a more generic FaaS solution, namely Firebase Cloud Functions.

Serverless computing is also applicable to more common medical transactional applications. A good example is the proposed system in [22] to support the delivery of drugs to Indian villages. Although its architecture is documented only briefly, the authors declare the implementation of backend services using AWS Lambda and Amazon API Gateway. Serverless computing can be of particular interest in event-driven data processing. This is undoubtedly the case in WBANS class systems (wireless body area networks). In their publication [23], the authors propose a system architecture for such applications, the backend part of which is located in the computing cloud. They compare the services that would be appropriate for the development of such a system from both the AWS and Azure clouds and ultimately decide on the first one using, inter alia, the AWS Lambda service. A solution also belonging to the WBAN class, implemented as a mobile application for monitoring the heart rate, presented in another paper [24], uses Google Cloud services, in particular Firebase Cloud Functions.

Last but not least, serverless computing can be applied for deep learning issues, which was presented in the example of medical services for patient-specific arrhythmia detection [25]. According to the authors, using AWS Lambda makes the serverless setup orders of magnitude cheaper and more scalable than a dedicated host running 24/7.

As can be seen from the above review of the state of research on the use of serverless computing in the field of healthcare, most applications concern cloud functions, which means they use services provided by a specific cloud provider (most often it is AWS Lambda). The authors of this paper decide to take a slightly different direction, namely to use a solution based on the Kubernetes cluster to be more in line with the cloud-agnostic approach. This is to avoid so-called vendor lock-in to a specific computing cloud and thus enable its easy implementation at various cloud computing providers, without the need for major changes in the proposed architecture of the solution.

3. Materials and Methods

The FaaS-based computing solutions for Kubernetes have been developed for many years. The most popular ones are Fission, Kubeless and Riff (based on Knative) [26], among other well-known products such as BlueNimble, FaaS, Fn, Funktion, Getsalt, IronFuncton, Kubeless, OpenWhisk, Nuclio and RainBond, and some of them have lived to see comparative analysis in the scholarly literature [27–31].

Among the solutions mentioned above, the Fission platform was selected. The reasons for this choice are solution maturity and production readiness, detailed documentation, ease of implementation and, above all, a high level of compliance with the core Kubernetes mechanisms.

Fission, for serving FaaS features, creates objects registered in the Kubernetes environment (so-called custom resource definitions, CRDs) and uses native Kubernetes resources as well. Although it provides a command line tool for managing its own environment (namely Fission CLI), it is possible and even more convenient to configure and manage the entire Fission using the standard Kubernetes API (e.g., using only a basic command line tool such as kubectl).

3.1. Fission—FaaS for Kubernetes

As has been mentioned, Fission is a Kubernetes-based framework, which enables the implementation of computing in line with FaaS concepts but on the basis of the K8s cluster, thus using all basic Kubernetes building blocks like *Image*, *Container*, *Pod*, *Service*, *Deployment*, *HPA*, *CRD*, etc.

The operating system, runtime libraries and domain modules of the given service are stored together as a so-called *Docker Image*. A *Container* is the entity hosting the processes being executed and can be instantiated from a *Docker Image*.

Kubernetes allows running so-called *Pods* which are the instances of at least one *Container* (and most commonly of exactly one *Container*). *Pods* are the smallest processing units being orchestrated by Kubernetes. To make it easier to handle the scaling of computation, *Pods* of the same type are grouped, and these groups are managed by *Deployments*. Services provided by running *Pods* can be exposed (internally in the Kubernetes cluster or externally) by Kubernetes *Service* resources, which define some network properties. The number of running *Pods* can be constant or variable, and in the latter case it can be controlled by Kubernetes resources, i.e., *HPA—Horizontal Pod Autoscaler*, which dynamically adjusts the number of *Pods* depending on the current load (measured by mean CPU utilization metric in pods).

To enable functional extensibility, Kubernetes allows one to define his/her own custom kinds of resources (so-called *CRD—Custom Resource Definition*). Although *CRDs* can extend the functionality of the whole environment, they are still natively managed by Kubernetes. That means that Kubernetes controls their lifecycles and makes them available through standard management mechanisms (common HTTP RESTful API and CLI tools).

3.2. Fission Architecture

Fission is based on a few custom resource definitions, and thus it is well integrated with Kubernetes. The main components of Fission are: the *Router*, *HTTP Trigger*, *Message Queue Trigger*, *Timer*, *Controller*, *Executor*, *Builder Manager*, *Builder Pod* and *StorageSvr*. The *Controller* is responsible for creating Fission CRD objects. The functions available through Fission can be invoked by the *Router*, *Message Queue Trigger* and *Timer* as a result of incoming HTTP requests, messages delivered to a queue or a topic occurrence of a time event (generated by a Kubernetes *Cron job*). Fission can work in two specific modes (https://fission.io/docs/architecture/executor/, accessed on 13 April 2022):

- *PoolManager*;
- *NewDeployment*.

When operating in *PoolManager* mode, functions are available as archive packages of source code (developed in Java, JavaScript, Go, etc.). The *Builder Manager* component uses the *Builder Pod* to fetch file archives from *StorageSvr* and builds executable artifacts and uploads them to a shared volume.

In response to some requests coming from the *Router*, the *Executor* should provide the address of the *Function* (Figure 1) which is exposed by some *Pod*.

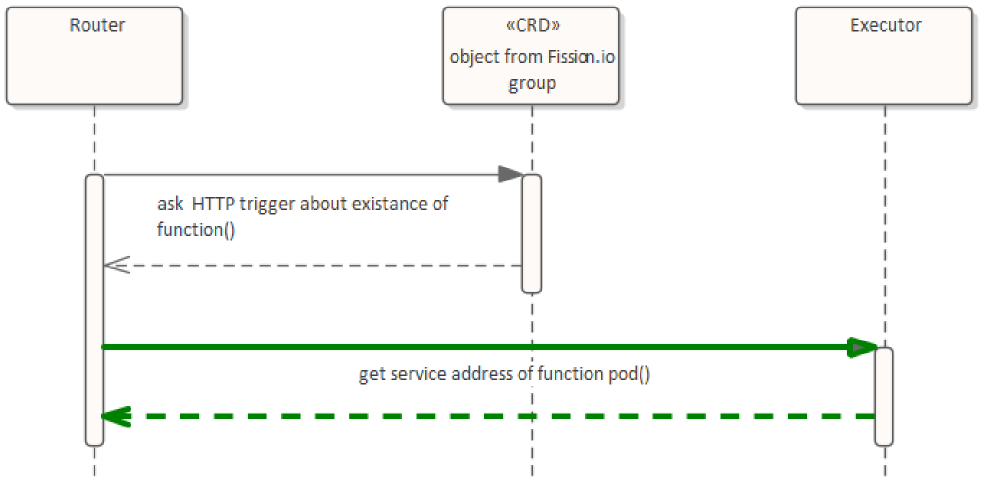

Router
«CRD»
object from Fission.io group
Executor
ask HTTP trigger about existance of function()
get service address of function pod()

Figure 1. Sequence Diagram: *Router* to *HTTP trigger* and *Router* to *Executor*.

Those *Pods* may be already instantiated/available (a warm start of the *Function*) or not (a cold start of the *Function*).

The *PoolManager* is responsible for holding a few so-called *Generic Pods*, whereas the *Executor* has to provide a *Function* to the *Router*. This process performed by the *Executor* is named the *specialization* of one of the *Generic Pods*. The *Executor* injects a proper package (the one previously prepared by the *Builder Pod*) into a running *Pod* and exposes it to the *Router*.

This mode (i.e., Pool Manager mode) allows having one poll of *Generic Pods* for the usage for any kind of *Function*, which is a big advantage. However, *Pods* should stay available all the time (causing a significant utilization of memory), and (what is more important) the process of *Pod* specialization takes a significant time (which may result in unacceptable timeouts for clients).

Those above-mentioned weaknesses can be overcome by using the *NewDeployment* mode. Here, the *Executor* spins up a dedicated *Pod* (or *Pods*) for the *Function* required by the *Router*. The *Executor* creates/applies a *Deployment*, *Pod*, *Service* and *HPA* for a needed *Function* (Figure 2). The number of *Pods* depends on *HPA* settings and can be adapted to a current load of *Function* calls.

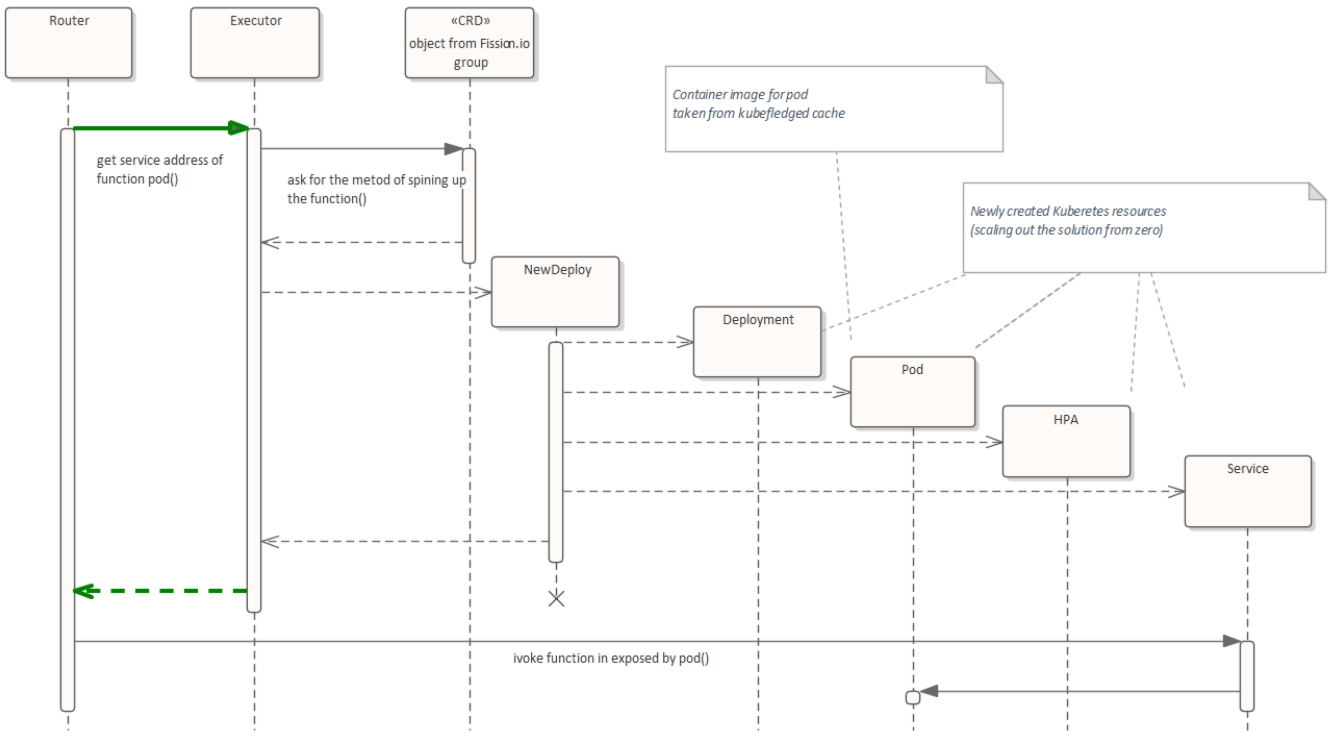

Router
Executor
«CRD» object from Fission.io group
Container image for pod taken from kubefledged cache
Newly created Kuberetes resources (scaling out the solution from zero)
get service address of function pod()
ask for the metod of spining up the function()
NewDeploy
Deployment
Pod
HPA
Service
ivoke function in exposed by pod()

Figure 2. Cooperation Diagram of *NewDeployment* mode of *Executor*—scaling out from zero *Pods*.

3.3. Using Fission in Cloud Healthcare Services

Pods may be scaled from the number of 0, which is especially important from the cost perspective. Because there is no *Pod* specialization (like in *PoolManager* mode), this mode requires preparing a separate *Docker Image* with the appropriate *Function* injected. For *NewDeployment* mode, *Pod* scaling out from 1 is fast (even faster than scaling out for *PoolManager* mode). However, *Pod* scaling out from zero (the instantiation of the first dedicated *Pod* from the dedicated *Docker Image*) is efficiently worse than *Pod* specialization (in *PoolManager* mode). To mitigate this weakness, a *Docker Image* cache mechanism was implemented. Due to applying the kube-fleged (https://github.com/senthilrch/kube-fledged, accessed on 13 April 2022) module, the images originally stored only in a remote Docker Image repository were also cached inside a Kubernetes cluster (on the persistent disks of the cluster nodes). That allowed for shortening the time of instantiating a new dedicated *Pod* for the required *Function* from the cached image significantly.

The functionalities of some parts of the system were refactored and adapted to work in line with the FaaS model. Previously, the modules were prepared as classic microservices developed for a standard JVM using Java and the spring-boot (https://spring.io/projects/

spring-boot, accessed on 13 April 2022) framework or using the Kotlin programming language and the Ktor (https://ktor.io, accessed on 13 April 2022) and Koin frameworks (https://insert-koin.io/, accessed on 13 April 2022).

Taking into account the requirement of lightness for *Function* processing (in a sense of desired low memory utilization) we decided to use a more memory-efficient technology stack, i.e., to use Graal VM (https://www.graalvm.org, accessed on 13 April 2022) (instead of a standard Java virtual machine) and the Qarkus framework (https://quarkus.io, accessed on 13 April 2022) (instead of spring-boot or Ktor).

The authentication and authorization services (important parts of the ACS—access control system [32]) and patient surveys of cloud healthcare services implemented for the FaaS processing model were deployed using Fission. These modules were redeveloped in the above-mentioned new technology stack (Graal VM + Quarkus).

4. Results

To confirm the advantages of applying the Fission solution to cloud healthcare services, experiments of two kinds were conducted:

(A) The execution of a single HTTP request (with a long time interval between subsequent executions);
(B) The application of a continuous variable load profile.

Both load profiles were used for testing the functionalities of authentication, authorization, survey defining and survey submitting.

The results of experiment (A) allow for characterizing the behavior of the system variants (classic and FaaS-based) of the system for scaling out from 0. The results of experiment (B) show the acting of the autoscaling mechanism for the classic variant and FaaS-based one of the system.

4.1. Results For Single-Request Experiments

The load profile (A) was generated by executing an HTTP request using the Postman (https://www.postman.com/use-cases/api-testing-automation/, accessed on 17 May 2022) tool. To describe the behavior of the system for such a test load, the authentication functionality was considered. The criteria, such as the mean time of request executions and the capacity of RAM used by a *Pod*, were taken into account. The experimental results for the authentication functionality are presented in Table 1. They allow for comparing the classic Kubernetes solution to the new one based on Fission with respect to the times of execution.

Table 1. Experimental results for authentication functionality—duration of request executions.

No	Classic ACS Microservice [ms]	FaaS ACS Warm Start [ms]	FaaS ACS Cold Start without Image in Cache [ms]	FaaS ACS Cold Start with Cached Image [ms]
1	52	57	7360	3140
2	48	56	3150	4160
3	50	60	6180	3150
4	51	52	3170	4150
5	50	58	4160	3240
6	53	55	12,360	4180
7	51	65	11,250	4150
8	55	54	3540	3170
9	50	56	4540	4160
10	54	53	12,740	3170
mean	51.4	56.6	6845	3665
std	2.12	3.78	3885.61	518.31

Applying Fission to cloud healthcare services allowed us to obtain the solution with improvements in many areas. First, the scale-out from 0 feature was successfully implemented, and it took only a few seconds for the cold start (mean 3.78 s) due to applying the image caching mechanism. Another benefit is the significant decrease in memory consumption. The volume of RAM used by standard *Pods* was 320 MB, while the dedicated Function Pods (Fission variant) consumed only 19.4 MB. In addition, the time of the warm start of service deployed as a function (56.6 ms) was very close in comparison to the response time of plain *Pods* (51.4 ms). The Student's t-test was used to confirm that there is no statistically significant difference between the means for the assumed confidence level of 0.95. All of that was achieved with compliance to Kubernetes technology and with the preservation of cloud provider agnosticism.

It is worth mentioning that the mean time of the cold start of FaaS ACS provided from the cached image was 6.8 s and was significantly lower than the mean time of instantiating the classic ACS microservice, which was 61 s.

Similar beneficial conclusions regarding execution times and memory consumption were obtained due to applying the FaaS approach based on Fission in implementing other aforementioned functionalities, i.e., authorization and defining/submitting patient surveys.

4.2. Results For Experiments with Continuous and Variable Load Profile

In experiment (B), the following load profile (trapezoid shape) was used:

$$\mathrm{NoR}(t) = \begin{cases} 0 & 0 \le t < t_1 \\ \frac{\mathrm{NoR}_{max}}{t_2 - t_1}(t - t_1) & t_1 \le t < t_2 \\ \mathrm{NoR}_{max} & t_2 \le t < t_3 \\ \frac{\mathrm{NoR}_{max}}{t_3 - t_4}(t - t_4) & t_3 \le t < t_4 \\ 0 & t_4 \le t \le t_5 \end{cases}$$

describing the dependency between the number of requests per second (NoR) and time. To generate the defined load profile, the Gatling (https://gatling.io/, accessed on 17 May 2022) tool was applied. A sample generated load for the following parameter values—$\mathrm{NoR}_{max} = 50$ rps, t_1 = 15 min, $t_2 = t_1 +$ 10 min, $t_3 = t_2 +$ 10 min, $t_4 = t_3 +$ 10 min and $t_5 = t_4 +$ 15 min = 1 h—is presented in Figure 3.

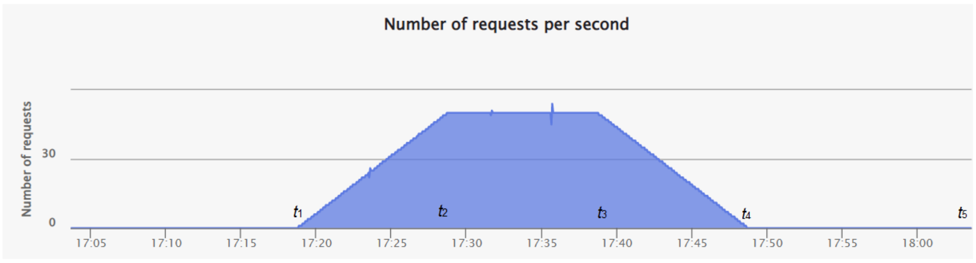


Figure 3. Sample trapezoid load profile—the course of the number of requests per minute.

The trapezoid test profile was used for load testing either classic Pods or FaaS ones. In the described solution, a classic ACS *Pod* has a value of RAM limit set to 1 GB. A FaaS ACS *Pod* has a value of RAM limit set to 64 MB. The classic solution is not scaled-in to 0. Moreover, it supports a high availability property (fault tolerance) so there are at least two *Pods* of ASC in a namespace. In the current version of the cloud healthcare service, 10 was taken (inherited assumption) as a value of the maximum number of classic *Pods* that can be instantiated by HPA during autoscaling. That means that the number of classic *Pods* may change from 2 to 10. That also means that the maximum for the sum of the RAM limits of running classic *Pods* is equal to 10 × 1 GB = 10 GB. Maintaining the assumption that the maximum memory limit for classic ACS *Pods* is 10 GB RAM, the maximum (for HPA) number of FaaS ASC *Pods* is 10 GB / 64 MB = 160.

The classic and FaaS solutions were compared using some quality indicator (QI) based on a total size of allocated memory measured as a sum of the limits of instantiated *Pods*. QI is an integrated sum of memory limits (SML) for all instantiated *Pods* along the time of the experiment divided by the time of the whole experiment (T_{max}):

$$\mathrm{QI} = \frac{1}{T_{max}} \int_0^{T_{max}} \mathrm{SML}(t)\, dt$$

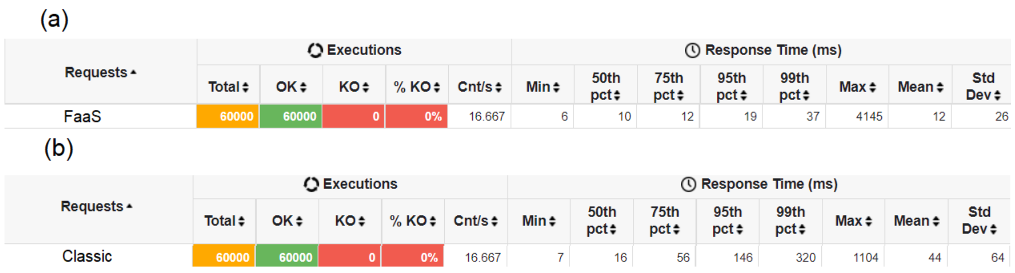
A similar quality indicator for estimating the cost of a system (but based on the integration of the number of processing units such as virtual machines or *Pods*) is used in [33].

The experiment consists of a series of one-hour loads defined by the described trapezoid test profile (NoR_{max} = 50 rps , $t_5 = 1$ h).

T_{max} was assumed to be 10 h, which means 10 times the usage of the one-hour load test profile.

In the following part of this subsection, the results of the load testing of authorization functionality based on the introduced quality indicator are presented.

The proposed method of evaluation used some constraints according to the assumed service level agreement (SLA). For the considered system, the SLA was based on a high order of quantile for the times of request execution. In accordance with the business clients of cloud healthcare services, it was assumed that the 95th-order quantile of execution times for an authentication request should be less than 200 ms. The resulting time statistics are presented in Figure 4a (FaaS solution) and Figure 4b (classic one).


(a)

Requests ▴	Executions					Response Time (ms)							
	Total ⬍	OK ⬍	KO ⬍	% KO ⬍	Cnt/s ⬍	Min ⬍	50th pct ⬍	75th pct ⬍	95th pct ⬍	99th pct ⬍	Max ⬍	Mean ⬍	Std Dev ⬍
FaaS	60000	60000	0	0%	16.667	6	10	12	19	37	4145	12	26

(b)

Requests ▴	Executions					Response Time (ms)							
	Total ⬍	OK ⬍	KO ⬍	% KO ⬍	Cnt/s ⬍	Min ⬍	50th pct ⬍	75th pct ⬍	95th pct ⬍	99th pct ⬍	Max ⬍	Mean ⬍	Std Dev ⬍
Classic	60000	60000	0	0%	16.667	7	16	56	146	320	1104	44	64

Figure 4. Statistics for times of request executions for FaaS solution (**a**) and classic one (**b**).

In addition, the distributions of the times of an execution are shown as histograms for both solutions in Figure 5a (FaaS solution) and Figure 5b (classic one).

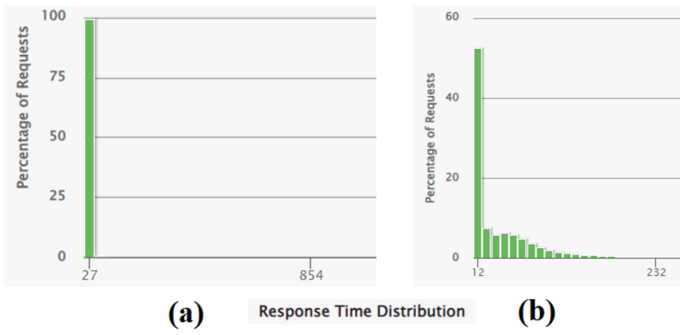


Figure 5. Histograms describing distributions of times of request executions (response times) for FaaS solution (**a**) and classic one (**b**).

We can confirm satisfying the mentioned SLA criterion by both solutions—both values of 95th columns are less than 200 ms in Figure 4.

Statistics such as means, medians and other quantiles are smaller for FaaS. In addition, the dispersion of the values of times (measured by standard deviation—Figure 4—or by the length of the support of the domain of histograms—Figure 5) is smaller for FaaS. All this indicates certain advantages of the FaaS solution over the classic one.

Empirically, we found the throughput limit defined as the maximum value of Cnt/s under two assumed constraints (the value of the 95th quantile of response times should be less than or equal to 200 ms and the fraction of successful responses should be equal to or greater than 95). For this purpose, we used the load range defined by the values of NoR$_{max}$ = 50, 60, 100 reqs/s. The resulting throughput limit for the classic version was 20 requests per second. For the considered load range, the two mentioned limits were always met for the FaaS variant (the limit was not reached). This makes the FaaS variant superior in terms of throughput limit.

Further observations about the courses of memory allocation are particularly important in terms of cost.

Sample courses of SML(t) during a one-hour interval for the FaaS solution and the classic one are presented in Figures 6 and 7. All courses were rendered using the user interface Grafana's (https://grafana.com, accessed on 17 May 2022) monitoring tool, which visualizes the monitoring data collected by Prometheus (https://prometheus.io, accessed on 17 May 2022).

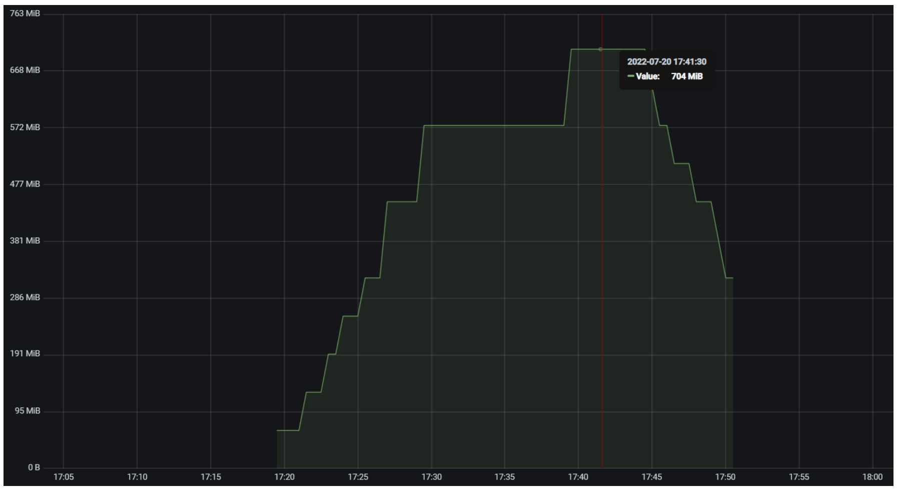


Figure 6. Sample course of sum of memory limits (SML) for FaaS solution during the one-hour experiment.

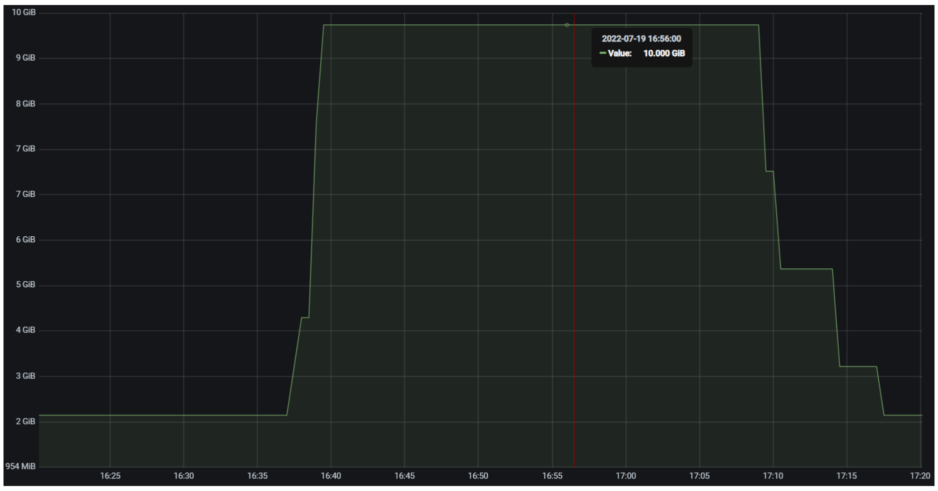


Figure 7. Sample course of sum of memory limits (SML) for classic solution during the one-hour experiment.

According to the knowledge about the memory limits (classic—1 GB ; FaaS—64 MB) we can observe:

- When the system is not under load (for example, when $0 < t < t_1$): two *Pods* are instantiated by the classic solution (2 GB of SML / 1 GB of memory limit = two instances);

- When the system is not under load: zero *Pods* are instantiated by the FaaS solution;
- When the system is under maximal load (for example, when $t_2 < t < t_3$): SML equals 10 GB for the classic solution;
- When the system is under maximal load: 10 *Pods* are instantiated by the classic solution (10 GB of SML/ 1 GB of limit = 10 instances). HPA is in a saturated state—the maximum number of *Pods* for the classic solution is achieved,
- When the system is under maximal load: SML equals only 704 MB for the FaaS solution;
- When the system is under maximal load: 11 *Pods* are instantiated by the FaaS solution (704 MB of SML / 64 MB of limit = 11 instances), which is a significantly smaller value than 160—the maximum number of *Pods* for HPA for the FaaS solution.

The final cost evaluation was based on the values of the quality indicator (OI) for the experiment lasting $T_{max} = 10$ h. The QI for the classic solution equaled 5944.3 MB. The QI for the FaaS solution equaled 264.9 MB which is a significant improvement (QI 22.5 times smaller) compared to the QI of the classic solution.

5. Conclusions

For many years now, cloud computing has filled the bulk of the computing infrastructure landscape for the public, research and commercial sectors. The advantages of this type of computing create immeasurable opportunities for the scaling, securing and instant deployment of services and thus make computing more common, cost-effective and less taxing on the environment. However, the mere fact of locating computing in the cloud does not yet guarantee the achievement of cost optimization. Cloud computing offers a whole range of services dedicated to different applications and different natures of workloads. Therefore, when deploying their own services, one should consider their load profile and adjust the system architecture and the use of cloud computing services of the appropriate type to it.

In the case of the case study described in this paper, the healthcare IT system consisted of a number of domain-specific services originally implemented in a homogeneous manner as plain pods deployed to the managed K8s cluster. However, we found that the production workload profiles of the various service groups differed greatly, making the use of cluster resources suboptimal. Particularly problematic was the medical survey service, which was not used at all most of the time, but there were unpredictable moments of increasing peaks of requests.

Ultimately, we decided to redevelop the cloud-based medical services in question using the Fission tool, and we noted a significant decrease in the RAM requirements of this part of the system. This allowed a more general finding to be drawn that there are services deployed to production with a specific workload profile for which the use of a FaaS approach can bring a reduction in the need for the cloud system infrastructure.

Among the other findings corroborated by the research conducted within the scope of this article, there are some interesting and particularly practically useful insights.

It is important to note that the resulting solution, which provides a lower consumption of computing resources, especially RAM, is at the same time a cloud-agnostic solution, which means that its use does not force the use of services specific to a particular cloud provider. This leads us to think about the reasonableness of the existence of FaaS-class solutions provided natively from the level of individual clouds (so-called cloud functions). Since a double benefit was achieved by implementing the Fission platform, can other benefits be achieved by relying on cloud functions (e.g., AWS Lambda)? It turns out that the creators of individual cloud functions prioritize ease of use in their solutions, while the implementation of the described functionalities based on Fission was not at all trivial either in terms of configuration or implementation, so it is precisely this type of additional cost that one must expect if one wants to remain independent of a specific cloud provider.

Another interesting observation is that it is sufficient to have a K8s cluster to implement the solution in question. It is worth noting that one of the most popular directions of cluster implementation is the building of the entire infrastructure of enterprises under the control of a single *control plane*—that is, within one logical K8s cluster. Having such an aggregation of computing resources (in the form of combining all computing nodes of the cluster into a single ecosystem) and then partitioning them vertically (e.g., different business areas) and horizontally (e.g., development, test, stage and production environments) allows for particularly efficient utilization. Implementing FaaS solutions based on a K8s cluster setup in such a model seems particularly reasonable.

Finally, it is worth noting that, despite achieving the highly desirable scale-down-to-zero option (since it practically does not allocate computing resources when the services in question cease to be used in a given period), it is possible to switch the services deployed based on Fission to an operating model based on classic *Pods* K8s with relatively little effort when their load profile becomes more regular and continuous.

The functional groups selected for implementation in the FaaS model in this case study were so sporadically used that the advantages of moving them to the FaaS model are indisputable. However, in the general case, it would be valuable to develop a general decision framework, taking into account the consumption profile of RAM, other computing resources and workload, to support the selection of individual microservices worth transforming to the FaaS model. This could be the subject of future authors' research.

Author Contributions: Conceptualization, D.R.A., Ł.W. and M.S.; methodology, D.R.A. and Ł.W.; software, D.R.A., Ł.W. and M.S.; validation, D.R.A. and Ł.W.; writing—original draft preparation, D.R.A. and Ł.W.; writing—review and editing, D.R.A. and Ł.W.; visualization, D.R.A. and M.S.; funding acquisition, D.R.A. and Ł.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by statutory research funds of the Department of Applied Informatics, Silesian University of Technology, Gliwice, Poland (02/100/BK22/0017).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACS	access control system
ADT	admit, discharge and transfer
AWS	Amazon Web Services
CDA	clinical document architecture
CDR	clinical data repository
CLI	command line interface
CRD	custom resource definition
ERP	enterprise resource planning
GCP	Google Cloud Platform
GKE	Google Kubernetes Engine
FaaS	Function-as-a-Service
HA	high availability
HPA	horizontal pod autoscaler
HIS	hospital information system
HL7	health level seven
IoMT	Internet of Medical Things
K8s	Kubernetes
PACS	picture archiving and communication system
RIS	radiology information system
S3	simple storage service
TCO	total cost of ownership

References

1. Hall, R.E.; Jones, C.I. The Value of Life and the Rise in Health Spending. *Q. J. Econ.* **2007**, *122*, 39–72. [CrossRef]
2. Griebel, L.; Prokosch, H.U.; Köpcke, F.; Toddenroth, D.; Christoph, J.; Leb, I.; Engel, I.; Sedlmayr, M. A scoping review of cloud computing in healthcare. *BMC Med. Inform. Decis. Mak.* **2015**, *15*, 17. [CrossRef]
3. Javadpour, A.; Wang, G.; Rezaei, S.; Chend, S. Power Curtailment in Cloud Environment Utilising Load Balancing Machine Allocation. In Proceedings of the 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), Guangzhou, China, 8–12 October 2018; pp. 1364–1370. [CrossRef]
4. Augustyn, D.R.; Wyciślik, Ł.; Sojka, M. The Cloud-Enabled Architecture of the Clinical Data Repository in Poland. *Sustainability* **2021**, *13*, 14050. [CrossRef]
5. Massaro, A. Information Technology Infrastructures Supporting Industry 5.0 Facilities. In *Electronics in Advanced Research Industries: Industry 4.0 to Industry 5.0 Advances*; Wiley: Hoboken, NJ, USA, 2022; pp. 51–101. [CrossRef]
6. Paul, S.; Riffat, M.; Yasir, A.; Mahim, M.N.; Sharnali, B.Y.; Naheen, I.T.; Rahman, A.; Kulkarni, A. Industry 4.0 Applications for Medical/Healthcare Services. *J. Sens. Actuator Netw.* **2021**, *10*, 43. [CrossRef]
7. Jiang, S.; Cao, J.; Wu, H.; Yang, Y.; Ma, M.; He, J. BlocHIE: A BLOCkchain-Based Platform for Healthcare Information Exchange. In Proceedings of the 2018 IEEE International Conference on Smart Computing (SMARTCOMP), Taormina, Italy, 18–20 June 2018; pp. 49–56. [CrossRef]
8. Haleem, A.; Javaid, M.; Singh, R.P.; Suman, R.; Rab, S. Blockchain technology applications in healthcare: An overview. *Int. J. Intell. Netw.* **2021**, *2*, 130–139. [CrossRef]
9. Faust, O.; Lei, N.; Chew, E.; Ciaccio, E.J.; Acharya, U.R. A Smart Service Platform for Cost Efficient Cardiac Health Monitoring. *Int. J. Environ. Res. Public Health* **2020**, *17*, 6313. [CrossRef] [PubMed]
10. Niu, X.; Kumanov, D.; Hung, L.H.; Lloyd, W.; Yeung, K. Leveraging serverless computing to improve performance for sequence comparison. In Proceedings of the BCB'19: 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, Niagara Falls, NY, USA, 7–10 September 2019; pp. 683–687. [CrossRef]
11. He, H.; Fu, S.; Wang, L.; Liu, S.; Wen, A.; Liu, H. MedTator: A serverless annotation tool for corpus development. *Bioinformatics* **2022**, *38*, 1776–1778. [CrossRef] [PubMed]
12. Crespo-Cepeda, R.; Agapito, G.; Vazquez-Poletti, J.; Cannataro, M. Challenges and opportunities of Amazon serverless lambda services in bioinformatics. In Proceedings of the BCB'19: 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, Niagara Falls, NY, USA, 7–10 September 2019; pp. 663–668. [CrossRef]
13. Grzesik, P.; Augustyn, D.R.; Wyciślik, Ł.; Mrozek, D. Serverless computing in omics data analysis and integration. *Briefings Bioinform.* **2021**, *23*, bbab349. [CrossRef]
14. Grzesik, P.; Mrozek, D. Serverless Nanopore Basecalling with AWS Lambda. In *Computational Science—ICCS 2021*; Paszynski, M., Kranzlmüller, D., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M.A., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 578–586.
15. Pérez, A.; Caballer, M.; Moltó, G.; Calatrava, A. A programming model and middleware for high throughput serverless computing applications. In Proceedings of the SAC'19: The 34th ACM/SIGAPP Symposium on Applied Computing, Limassol, Cyprus, 8–12 April 2019; Volume Part F147772, pp. 106–113. [CrossRef]
16. Nastic, S.; Rausch, T.; Scekic, O.; Dustdar, S.; Gusev, M.; Koteska, B.; Kostoska, M.; Jakimovski, B.; Ristov, S.; Prodan, R. A serverless real-time data analytics platform for edge computing. *IEEE Internet Comput.* **2017**, *21*, 64–71. [CrossRef]
17. Carrizales-Espinoza, D.; Sanchez-Gallegos, D.; Gonzalez-Compean, J.; Carretero, J.; Marcelin-Jimenez, R. SeRSS: A storage mesh architecture to build serverless reliable storage services. In Proceedings of the 2022 30th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), Valladolid, Spain, 9–11 March 2022; pp. 88–91. [CrossRef]
18. Ergüzen, A.; ünver, M. Developing a file system structure to solve healthy big data storage and archiving problems using a distributed file system. *Appl. Sci.* **2018**, *8*, 913. [CrossRef]
19. Sadek, J.; Craig, D.; Trenell, M. Design and Implementation of Medical Searching System Based on Microservices and Serverless Architectures. *Procedia Comput. Sci.* **2021**, *196*, 615–622. [CrossRef]
20. Shaik, F.; Khalid, A.; Ismail, H. Covisstance chatbot. In Proceedings of the ArabWIC 2021: The 7th Annual International Conference on Arab Women in Computing in Conjunction with the 2nd Forum of Women in Research, Sharjah, United Arab Emirates, 25–26 August 2021. [CrossRef]
21. Bharti, U.; Bajaj, D.; Batra, H.; Lalit, S.; Lalit, S.; Gangwani, A. Medbot: Conversational artificial intelligence powered chatbot for delivering tele-health after COVID-19. In Proceedings of the 2020 5th International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 10–12 June 2020; pp. 870–875. [CrossRef]
22. Chinchole, S.; Kulkarni, A.; Matai, L.; Kotadiya, C. A real-time cloud-based messaging system for delivering medication to the rural areas. In Proceedings of the 2017 International Conference on Intelligent Sustainable Systems (ICISS), Palladam, India, 7–8 December 2018; pp. 475–479. [CrossRef]
23. Paul, P.; Loane, J.; McCaffery, F.; Regan, G. A serverless architecture for wireless body area network applications. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Proceedings of the Model-Based Safety and Assessment: 6th International Symposium, IMBSA 2019, Thessaloniki, Greece, 16–18 October 2019*; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11842 LNCS, pp. 239–254. [CrossRef]

24. Pandey, S.; Hicks, D.; Goyal, A.; Gaurav, D.; Tiwari, S. Mobile notification system for blood pressure and heartbeat anomaly detection. *J. Web Eng.* **2020**, *19*, 747–773. [CrossRef]
25. Marefat, M.; Juneja, A. Serverless data parallelization for training and retraining of deep learning architecture in patient-specific arrhythmia detection. In Proceedings of the 2019 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI), Chicago, IL, USA, 19–22 May 2019. [CrossRef]
26. Sayfan, G. *Mastering Kubernetes: Automating Container Deployment and Management/Gigi Sayfan*, 3rd ed.; Packt: Birmingham, UK, 2020.
27. Mohanty, S. Evaluation of Serverless Computing Frameworks Based on Kubernetes. Master's Thesis, Aalto University, School of Science, Singapore, 2018.
28. Mohanty, S.K.; Premsankar, G.; di Francesco, M. An Evaluation of Open Source Serverless Computing Frameworks. In Proceedings of the 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Nicosia, Cyprus, 10–13 December 2018; pp. 115–120. [CrossRef]
29. Kritikos, K.; Skrzypek, P. A Review of Serverless Frameworks. In Proceedings of the 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, Switzerland, 17–20 December 2018; pp. 161–168. [CrossRef]
30. Palade, A.; Kazmi, A.; Clarke, S. An Evaluation of Open Source Serverless Computing Frameworks Support at the Edge. In Proceedings of the 2019 IEEE World Congress on Services (SERVICES), Milan, Italy, 8–13 July 2019; Volume 2642, pp. 206–211. [CrossRef]
31. Massaro, A.; Galiano, A.; Scarafile, D.; Vacca, A.; Frassanito, A.; Melaccio, A.; Solimando, A.; Ria, R.; Calamita, G.; Bonomo, M.; et al. Telemedicine DSS-AI Multi Level Platform for Monoclonal Gammopathy Assistance. In Proceedings of the 2020 IEEE International Symposium on Medical Measurements and Applications (MeMeA), Bari, Italy, 1 June–1 July 2020; pp. 1–5. [CrossRef]
32. Wyciślik, Ł. Access control system based on generalization of rbac model-concept, architecture and sample implementation. In Proceedings of the Technologie Przetwarzania Danych: TPD 2010. III Konferencja Naukowa, Materiały Konferencyjne, Poznań, Poland, 21–23 June 2010; Wydawnictwa Naukowo-Techniczne: Warszawa, Poland, 2010.
33. Augustyn, D.R.; Warchal, L. Metrics-Based Auto Scaling Module for Amazon Web Services Cloud Platform. In *Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation*; Kozielski, S., Mrozek, D., Kasprowski, P., Małysiak-Mrozek, B., Kostrzewa, D., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 42–52.