Transition from Serverfull to Serverless Architecture in Cloud-Based Software Applications

Chapter · December 2020

DOI: 10.1007/978-3-030-63322-6_24

4 authors:

Oliviu Matei
Universitatea Tehnica Cluj-Napoca
99 PUBLICATIONS **787** CITATIONS
SEE PROFILE

Pawel Skrzypek
AI Investments
18 PUBLICATIONS **144** CITATIONS
SEE PROFILE

Robert Heb
5 PUBLICATIONS **35** CITATIONS
SEE PROFILE

Alexandru Moga
4 PUBLICATIONS **41** CITATIONS
SEE PROFILE

Transition from Serverfull to Serverless Architecture in Cloud-Based Software Applications

Oliviu Matei[1], Pawel Skrzypek[2], Robert Heb[3] and Alexandru Moga[3]

[1] Dept. of Electrical Engineering, North University of Baia Mare,
Str. V. Babes, 430083, Baia Mare, Romania
oliviu.matei@cunbm.utcluj.ro
[2] 7bulls, Warszaw, Poland
pskrzypek@7bulls.com
[3] HOLISUN, Baia Mare, Romania
robert.heb@holisun.com, alexandru.moga@holisun.com

Abstract. This paper makes a practical comparison between serverfull and serverless architectures for the same specific cloud application used for face recognition. It turns out that both approaches have advantages and disadvantages and their specific use needs to be carefully assessed in the design phase of the software life cycle.

Keywords: cloud, serverless computing, software architecture.

1 Introduction

In the sphere of cloud world, a new paradigm arises, namely serverless computing, which is an execution model based on small components which start or die based on their needs and usage. In such case, the cloud resources are dynamically allocated. Pricing is based on the actual amount of resources consumed by an application, rather than on pre-purchased units of capacity [1].

In many specific application, serverless computing is the optimal solution which allows very easily scaling, capacity planning and maintenance of the application. No full servers are needed anymore and the resources are not stuck for ever for one platform. [2].

This should not be confused with computing or networking models that do not require an actual server to function, such as peer-to-peer (P2P).

Almost all serverless vendors provide function as a service (FaaS) platforms, which execute application logic but do not store data. In 2008, Google released Google App Engine, which featured metered billing for applications that used a custom Python framework, but could not execute arbitrary code [3].

Kubeless and Fission are two Open Source FaaS platforms which run with Kubernetes [4]. The first public cloud vendor providing serverless facilities was Amazon with its AWS Lambda, in 2014 [5]. Several additional AWS serverless

tools such as AWS Serverless Application Model (AWS SAM) Amazon CloudWatch, and others accompany the serverless service. Google Cloud Platform offers Google Cloud Functions since 2016 [6]. IBM provides IBM Cloud Functions in the public IBM Cloud since 2016 [7]. Microsoft Azure offers Azure Functions, either in the Azure public cloud or on-premises via Azure Stack [8].

2 Functionizer project

The main goal of the project is to create a unique platform that will optimize and manage the deployment of serverless applications in multi-cloud environment. This approach is an outcome of a growing market of data-intensive applications that constantly pursue better resource- and cost-efficiency. Serverless computing is positioned to change the state-of-play for cloud applications.

The business case must validate the Functionizer platform [9], by demonstrating that serverless computing can be effectively incorporated in the multi-cloud domain and demonstrate how Functionizer makes deployment and management of multi-cloud data-intensive applications faster, simpler and cheaper. The business case focuses on a certain software solution, which takes an audio/video streaming from wearable devices (namely smart glasses) and processes it on a server for face or image recognition. It has applications is several fields, such as:

- **Industry**. As a company there are situations that require a specialist to be present at various interventions. You can ship a pair of glasses anywhere you need and have an employee ready to be equipped with them. The glasses will transmit a live feed of whatever the employee is watching, back to a support center where your expert will be able to provide the much needed assistance.
- **Medicine and Emergency response**. Doctors and paramedics can be coordinated by a specialist from the Emergency Room during resuscitation maneuvers.
- **Training**. The lecturer can perform live demonstrations (presenting equipment, performing surgery or health and safety), while students can see exactly what he is doing in real time, classes can be recorded and all that while the lecturers are using both their hands.

The features employed by the solutions include:

- **Audio/video streaming**: The engineer and the technician can talk to each other and see what the other sees. The stream is encoded using H.264 encoder. If the device supports hardware acceleration, this feature is also used for better video quality. The default frame rate is 30 fps, and the default image resolution is 640x480. They resolution may vary between 320x240 and 1028x768, depending on the bandwidth. The unidirectional stream bandwidth is 1 Mb/s at a resolution of 640x480. As the communication is bidirectional, the reasonably needed bandwidth is 2 Mb. If there is a multi-user conference with n users, the bandwidth should be 2xn Mb.

- **Hands free**: While the technician has the support of the engineer and streaming back what he sees, he also has his hands free and available for using the tools he needs. Therefore, the efficiency is not affected in any way.
- **Chat**: This can be used when the audio stream is bad, or the user needs to send a model or serial number. The technician can send the inventory codes of the assets or their models or serial numbers. Moreover, the chat is the base communication means for drawings and file transfer. Of course, they are not visible in the chat form.
- **Drawings**: The engineer can draw simple shapes, such as rectangles, circles, polylines, and lines for spotting points of interest and making himself clearer in designating things. More complex shapes are useless, because if the glasses wearer moves her head, the shapes may not focus on the desired spot.
- **File transfer**: The engineer can send maps of the pipelines, maintenance manuals, and designs of the specific equipment or pipes to be repaired directly to the technician. The file can be opened on the glasses if there is a viewer installed. The most often transferred files are AutoCAD, pdf, images, doc, and txt files.
- **Platform independency**: We tested on the following OS: Android 5.1+, ReticleOS, iOS, and Windows. The devices used are desktops/notebooks, smart glasses (ODG R7 (HL) and Epson Moverio BT-300), and smart phones with the indicated OS. This means that the engineer can provide his support even when he is mobile (whether using Android or iPhone).
- **Snapshots**: These are possible from the support center (desktop/notebook) for documenting interventions. If a solution is not available because of very specific configuration of the pipe or equipment, the engineer can take a snapshot and further research the possible solutions.
- **Recording**: This is similar to snapshots. The recordings are stored when they are ended. This means that during a teleconference there can be more recordings (only of the important phases). The engineer can decide which parts of the conversation are worth being recorded and later stored. This improves the quantity of institutional knowledge in the company, which usually is lost when a technician leaves the company.
- **Multi-user video-conferencing**: This is an option if they are all available from the beginning. For each user, some basic information (e.g. GPS coordinate and the browser) is available.

3 Serverfull architecture

The serverfull architecture of AR Assistance follows client-server and is described in figure 1. The client side is accessed by the user and consists of smart glasses, physically. The server side is, of course, more complex and assumes all functionalities already provided by AR Assistance as well as the ones to be implemented during the Functionizer project.

In the architecture in figure 1, different kinds of application components are represented with different colours:

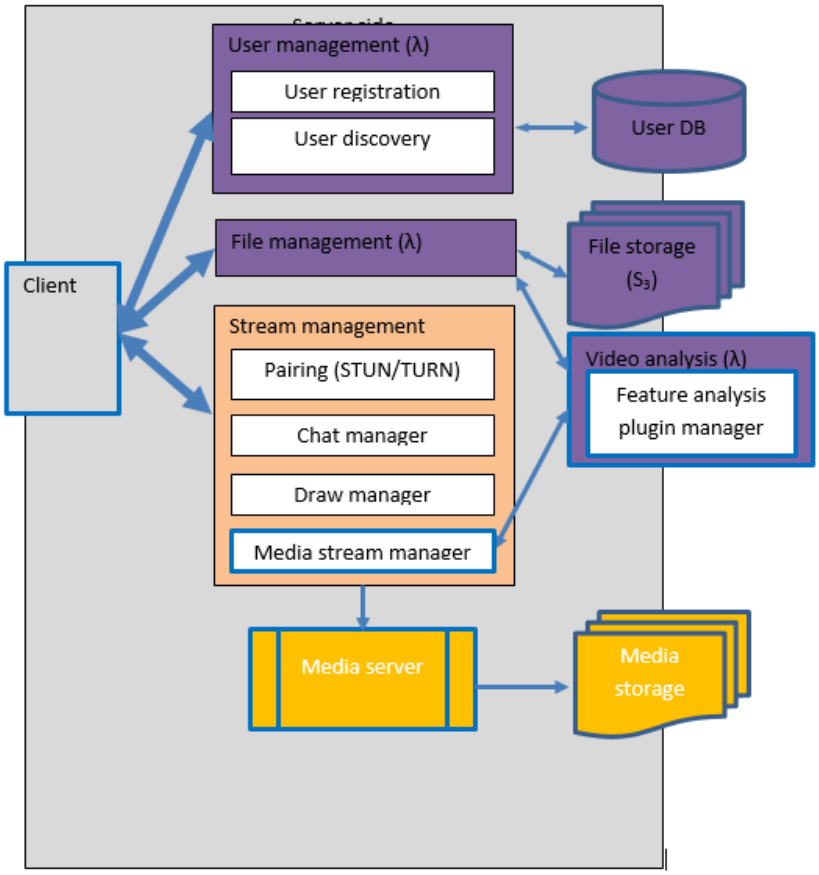

Fig. 1. The serverfull architecture of the application

– The gray components are the two tiers of the application – the client and the server.
– The green components are already implemented in a classical manner (i.e., statefully).
– The mauve components are to be deployed in the cloud (e.g. AWS).
– The ochre components represent the third party modules used within the development process.

The server-side modules include:

– User management – needed for managing the users (add, remove, modify and grant rights), their related info and meta-data.
– File management – needed for storing the files to be transferred within the application.
– Stream management offers the largest and most complex functionalities, related to the management of audio-video streams, such as:

- Pairing the ends of the conversation (two or more clients);
- Video streaming;
- Sending text from one end to the other;
- Sending graphic info from one end to the other.

3.1 Sequence diagram of the serverless approach

The sequence diagram of the serverful approach is depicted in figure 2.

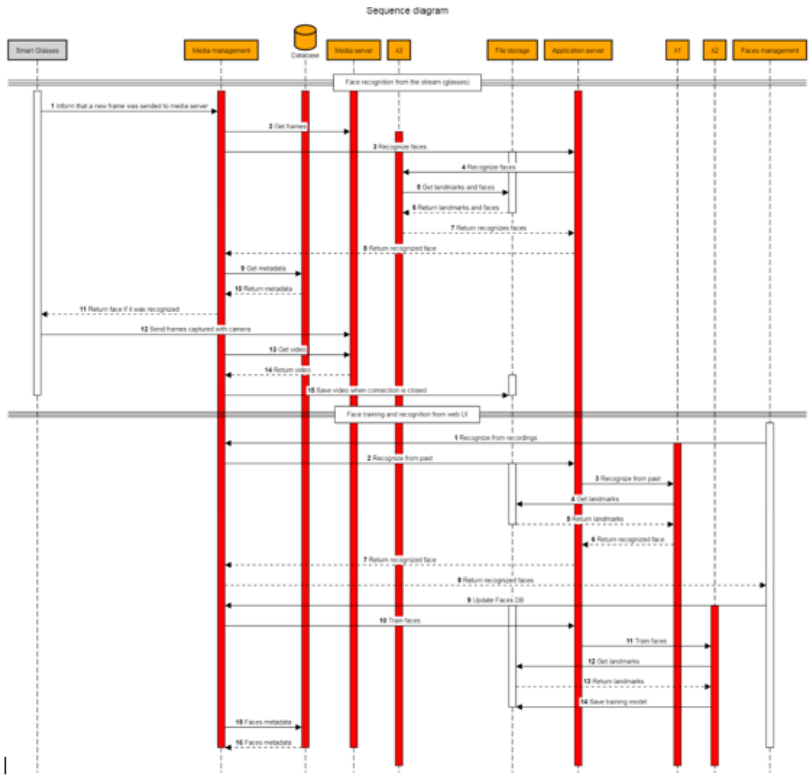

Fig. 2. The sequence diagram of the serverfull approach

Besides the modules already provisioned in the serverless approach and depicted in Figure 8, this serverfull exhibits some extras:

– The Application server, containing the specific components (λ_1, λ_2 and λ_3) modules;
– All Application server, λ_1, λ_2 and λ_3 are on forever;
– All calls of λ_1, λ_2 and λ_3 pass through the Application server.

3.2 Serverless architecture

The image/face recognition related modules are very complex, therefore we present the second level decomposition of AR Assistance architecture related to them and are depicted in figure 3.

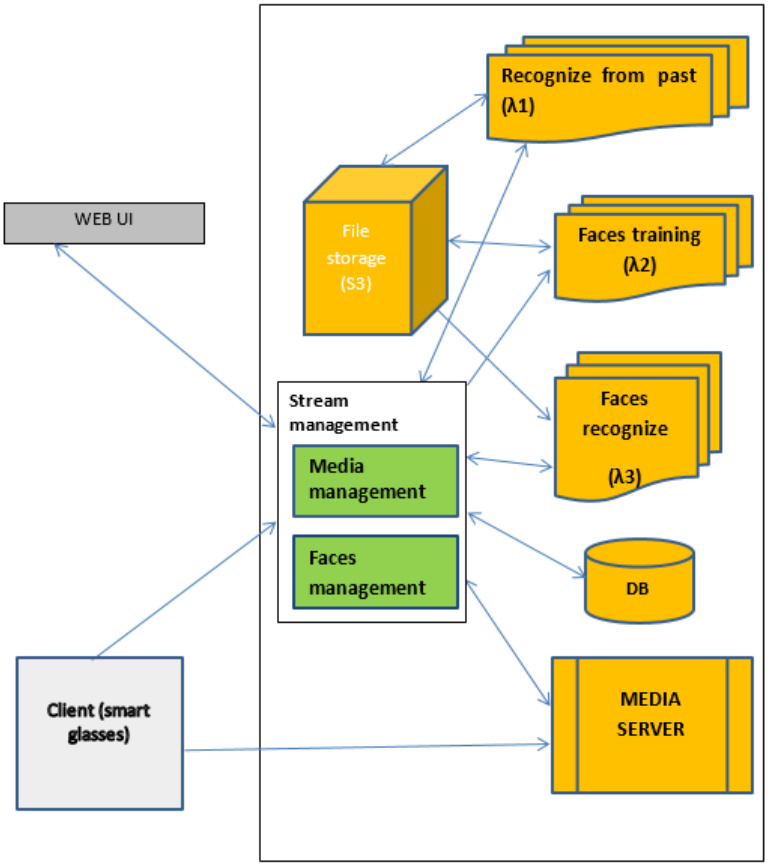

Fig. 3. The serverless architecture of the application

The colours of the components have the following meanings:

- The gray components represent the clients (dark grey is the web client and the light grey depicts the wearable client).
- The green components are already serverfull components.
- The ochre components cloud side components.

The architecture consists of:

- 1st tier (the client)
- 2nd tier (the business logic) consisting of:
  - Serverful components are deployed in the cloud and refer to:
    - Stream management
    - server (Kurento Media Server)
    - Database (DB) (Maria DB)
  - Serverless components:
    - 3 lambda functions: Recognize from past, Faces training and Faces recognize
    - 3rd party component for file storage (e.g. Amazon S3 service).

3.3 Sequence diagram

The sequence diagram for AR Assistance is depicted in figure 4. The red life lines are serverful (belonging to components living indeterminable), whereas the green life lines belong to serverless components and are usually relatively short.

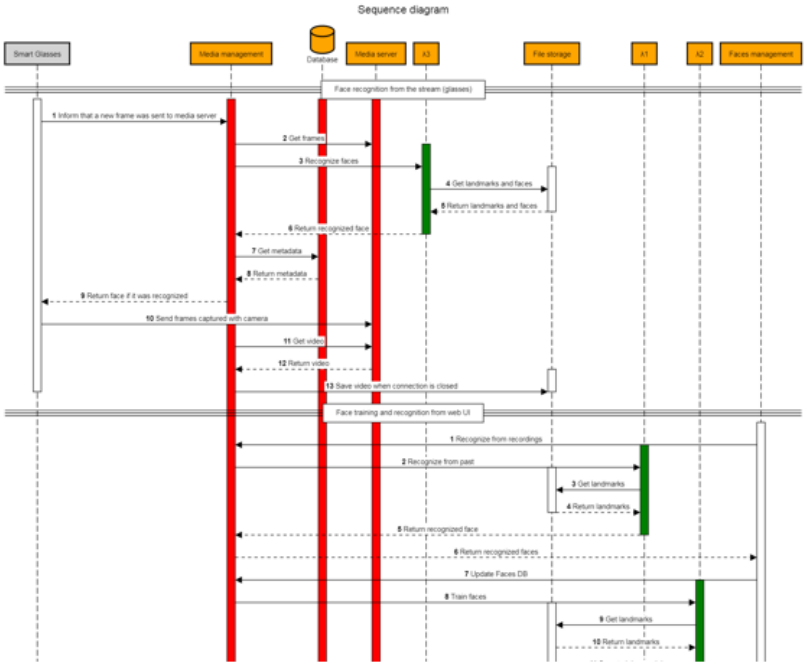

Fig. 4. The serverless sequence diagram

The sequence diagram depicted in Figure 4 has two lines - online (from the wearable client) and off-line (from the web client), split with a horizontal bar. All use cases start from the client - web for training, respectively wearable for production.

- The wearable client connects to media manager and establishes a communication session. In turn, media manager captures the frames from the Media Server (which is connected to the wearable via WebRTC) and transfers them to λ_3 (for face recognition), which returns the metadata of the recognized person (if it is the case) to Media Manager and from there to the client. For face recognitions, λ_3 uses the images stored in File Storage.
- In the same time, the video stream stored for further detailed processing and historical augmentation.
- Off-line, the face recognitions can be run on stored videos. The action is initiated from the web client.
- However, the face recognition algorithm needs to be trained for running properly. This sequence has been depicted as last one because it is less important in the economy of the application (as it runs less than the former ones).

Communication modes The underlying communication mechanisms of the sequence diagram in figure 4 are depicted in figure 5. The client invokes the 2nd tier by socket connections. The serverless components are called using a specific protocol by an SDK specific for each cloud.

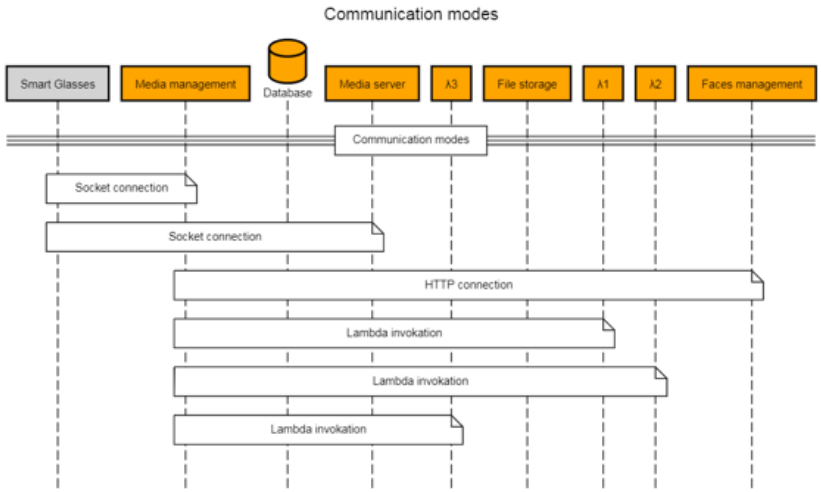

Fig. 5. The communication modes of the serverless approach

AWS Lambda supports synchronous and asynchronous invocation of a Lambda function, called *on-demand invocation* [10]. When the AWS service is used as a trigger, the invocation type is predetermined for each service.

4 Discussions

In many cases, applications comprise application servers which could orchestrate the execution of certain application functionalities offered as a service usually within the same host as the application server. We advocate that through serverless computing, it is possible to either totally remove the application server (e.g., when it just plays the role of a proxy) or to replace it with a light serverless function compositor (e.g., when it does involve some orchestration logic). Through this transformation, it can then be possible to reduce costs as:

- the serverless components live only when executed;
- they map to a smaller operation cost;
- they can be elastically scaled in quite high instance numbers in contrast to the case of application servers (and respective services) which might be restrained based on the elasticity bounds enforced by the hosting cloud providers.

Table 1 sumarizes the comparison between the serverfull and serverless architectures or AR Assistance.

Table 1. Comparison between serverfull and serverless architectures

Serverfull	**Serverless**
Architecturally	
The application server, which hosts and controls the specific components is architecturally defined	The serverless components are dynamically created on demand and very atomic. That allows a large flexibility of deployment.
Interaction and lifetime	
The calls of the specific components pass through the application server. The application server along with the specific components live indefinitely long	The calls of the specific components go directly to them with no imposed proxies. The specific components live only as long as they are running
Deployment	
The components, hosted by the application server, run on the same infrastructure.	The serverless components may run in various clouds (various infrastructures).

5 Conclusions

Serverless computing is a new concept more and more used in the cloud. It has its own advantages and disadvantages.

The advantages consist of:

- Serverless can be more cost efficient than renting full underused servers. There is significant research on optimizing the cost function in multi-cloud deployment [11–13].

- Serverless architecture are very flexible and very scalable as they use only needed resources, no extra resources being allocated apriori.
- Serverless components are architecturally simple and the developer does not have to worry about multithreading or directly handling HTTP requests in their code.
- Serverless paradigm is very suitable in the context of Internet of Things [14, 15].

On the other hand, there are also associated disadvantages:

- The serverless code infrequently used may show greater response latency until is starts, comparing with a serverfull .
- Serverless computing is not suited for resource intensive computation efforts, e.g. high-performance computing, because it would likely be cheaper to use servers.
- Serverless architectures lack of deep debugging and monitoring facilities.
- The amount of code vulnerable at cyber attacks is significantly larger compared to traditional architectures.
- The portability of serverless components may be an issue when changing the cloud providers, although it is covered by International Data Center Authority (IDCA) in their Framework AE360.

Acknowledgement

This work has received funding from the Functionizer Eurostars project and the EUs Horizon 2020 research and innovation programme under Grant Agreement No. 731664.

References

1. Miller, Ron. "AWS Lambda makes serverless applications a reality." TechCrunch. Retrieved 10 (2016).
2. Raines, Geoffrey, and Lawrence Pizette. "Platform as a service: A 2010 marketplace analysis." MITRE Technicel Paper (2010).
3. Zahariev, Alexander. "Google app engine." Helsinki University of Technology (2009): 1-5.
4. Brewer, Eric A. "Kubernetes and the path to cloud native." Proceedings of the Sixth ACM Symposium on Cloud Computing. 2015.
5. Villamizar, Mario, et al. "Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures." 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). IEEE, 2016.
6. Lynn, Theo, et al. "A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms." 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, 2017.
7. Cash, S., et al. "Managed infrastructure with IBM cloud OpenStack services." IBM Journal of Research and Development 60.2-3 (2016): 6-1.

8. Sreeram, Praveen Kumar. Azure Serverless Computing Cookbook. Packt Publishing Ltd, 2017.
9. Kritikos, Kyriakos, et al. "Towards the Modelling of Hybrid Cloud Applications." 2019 IEEE 12th International Conference on Cloud Computing (CLOUD). IEEE, 2019.
10. Kiran, Mariam, Peter Murphy, Inder Monga, Jon Dugan, and Sartaj Singh Baveja. "Lambda architecture for cost-effective batch and speed big data processing." In 2015 IEEE International Conference on Big Data (Big Data), pp. 2785-2792. IEEE, 2015.
11. G. Horn and P. Skrzypek, "MELODIC: Utility Based Cross Cloud Deployment Optimisation," 2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA), Krakow, 2018, pp. 360-367, doi: 10.1109/WAINA.2018.00112.
12. G. Horn and M. Rózańska, "Affine Scalarization of Two-Dimensional Utility Using the Pareto Front," 2019 IEEE International Conference on Autonomic Computing (ICAC), Umea, Sweden, 2019, pp. 147-156, doi: 10.1109/ICAC.2019.00026.
13. Matei, Oliviu. Evolutionary computation: principles and practices. Risoprint, 2008.
14. Matei, Oliviu, et al. "Collaborative data mining for intelligent home appliances." Working Conference on Virtual Enterprises. Springer, Cham, 2016.
15. Di Orio, Giovanni, et al. "A platform to support the product servitization." Int. J. Adv. Comput. Sci. Appl. IJACSA 7.2 (2016).