An Introduction to Serverless Computing

Christian Bargmann
christian.bargmann@haw-hamburg.de
Hamburg University of Applied Sciences
Hamburg, Germany

ABSTRACT

Initially launched as an isolated solution by Amazon in 2014 and today known as AWS Lambda [1], the Serverless paradigm is now supported by all major cloud providers. Whether AWS, Google, Microsoft or IBM, the idea is always the same. Serverless Computing allows developers to run applications and services in the cloud without having to worry about server provisioning, scaling and management. The products that are often provided as "*Function-as-a-Service*" (FaaS) are short-lived runtime environments that are dynamically assigned and scaled by the provider. This work is intended to provide an introduction to Serverless Computing. To begin with, terms are explained in this context. Then Serverless architectures and designs are described and different scenarios for Serverless Computing are presented. Thereafter different challenges and opportunities of Serverless Computing are discussed.

KEYWORDS

Serverless Computing, FaaS, Cloud Computing, Software Architecture

1 SERVERLESS COMPUTING

Serverless Computing is a cloud computing paradigm where the cloud provider provides a runtime environment for executing server-side logic. This server-side logic is implemented by short-lived, stateless functions called *Serverless functions*. The execution environment for these functions is called *Function-as-a-Service*. Serverless Computing is based on the idea of event-driven application architecture. This application architecture originates from the current trend of architectures consisting of containers and microservices in cloud environments. As a result, functions are executed in a Serverless application in response to the triggering of certain events [4]. When a Serverless function is triggered, the cloud provider dynamically allocates the computing capacity required to execute the function. After the execution is completed, the computing capacities are released again. Unlike the original Cloud Computing offerings, Serverless Computing does not require the reservation of computing capacity in advance. Previous cloud computing offerings required this, often resulting in problems due to misjudgement of server utilization. Reserving too much computing capacity resulted in high costs for unused resources. If, on the other hand, the server utilization was underestimated and too little computing capacity was reserved, high response times resulted [10] [26].

As the name suggests, Serverless Computing does not directly use servers. The term is often criticized because although the word Serverless suggests the absence of servers, they are still in use. In the form of a physical host or virtual machine, however, they work in the background to execute program code [24]. They are hidden by an abstraction layer from the user of a Serverless platform. If Serverless is nothing more than another abstraction layer on existing resources or platforms, one can quickly come to the conclusion that Serverless is nothing more than Platform as a Service (PaaS). This is not wrong, because Serverless in the end is based on a platform managed by the cloud provider and offered as a service. But Serverless is to be seen as a logical evolution of PaaS and thus a special form of this environment.

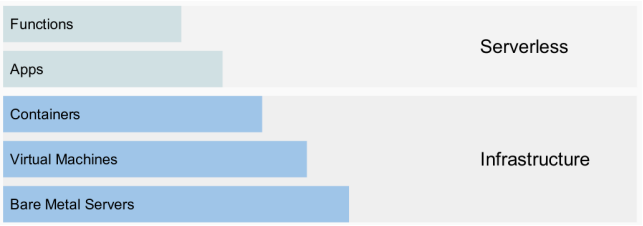

Figure 1: Representation of resources that fall within the area of responsibility of Serverless Computing. [5]

There are two main differences between classic PaaS and Serverless. First, in a Serverless world, the artifacts and resources that are deployed are much smaller. For example, not applications or containers are deployed anymore but only individual resources the size of a single function. On the other hand, with a PaaS, the developer still has to take care of availability and scaling himself. He must configure, monitor and adapt them. With Serverless, this happens implicitly by the amount of requests. Over- and under-provisioning of the environment is therefore no longer a danger, as only those resources are allocated that are really needed. The billing

Grundseminar M.Sc., February 28, 2019, Hamburg, Germany
2019.

model for Serverless Computing is also adapted to this. Only what has actually been used is paid for and idle times of a system is not billed [7].

From a developer's point of view, Serverless means that the time-consuming installation and management of server capacities is no longer necessary. This was previously necessary to get an application up and running. Instead, a provider's cloud service in the form of a "black box" is used. If the developer uploads program code to the Serverless platform, the cloud provider takes over the complete administration of the required server infrastructure, including server and operating system maintenance. The developer of the Serverless application has no control over the computing capacities, their allocation and scaling and only has to implement the Serverless functions. The cloud provider is responsible for provisioning, monitoring, maintaining, scaling and fault tolerance of the resources. The creation of prototypes is significantly accelerated by the addition of an abstraction level and without the need to allocate computing capacities for the execution of functions. Through the event-based architecture of a Serverless application and the billing according to the actual computing capacity used, the Serverless Computing approach brings advantages over conventional cloud computing paradigms in various areas. This applies, for example, to cloud applications in the Internet of Things area, where connected devices such as sensors and actuators sporadically output data. In response, Serverless functions can be executed. The Serverless Computing approach is also suitable for web applications with lightweight backend functions. Furthermore, Serverless functions are suitable as connectors between different services of a cloud provider, which serve as events. Since the Serverless platform controls the allocation of computing capacity for the execution of Serverless functions, it also guarantees scaling according to the requests.

2 SERVERLESS ARCHITECTURES

Serverless architectures offer new possibilities for building cloud applications. The key benefits of Serverless architectures include minimal application deployment and operations, but also a highly scalable and fault tolerant environment. Event-driven architectures are ideal for Serverless functionality because they can loosely couple different components together.

Serverless architectures draw a clear line between the code actually executed and its hosting environment. Serverless architectures are event-driven, i.e. functions are defined that are subsequently invoked by a trigger. After a function has been executed, the allocated resources are released again immediately. A trigger for a function can come from different sources, e.g. a simple webhook, a cronjob or a file upload. The activation of the trigger finally leads to the execution of the code defined in the function. Many cloud providers offer pre-defined APIs for triggers to allow Serverless functions deployed on the platform to respond to specific events. Unlike containers, for example, Serverless functions have a lifetime of a few milliseconds to a few minutes [9] [17].

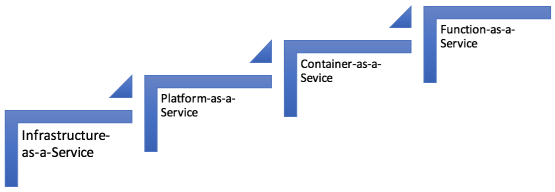

Figure 2: Serverless Computing or Function-as-a-Service is the fourth way to consume cloud computing. [11]

Serverless functions can be implemented both synchronously and asynchronously. A typical use case for the synchronous model is a Restful API endpoint where an incoming HTTPS request triggers a Serverless function. The caller waits until the function returns a result. In an asynchronous scenario, the caller does not wait until the function returns a result. Socket notifications or push notifications to a mobile device can be used to provide asynchronous feedback to the user. Serverless functions can be linked together to form an event system and realize more complex processing chains [21].

2.1 Serverless Architecture Designs

Serverless functions can be combined in different ways to form a comprehensive software architecture that benefits from the advantages of Serverless Computing. The following are some of the commonly used designs.

2.1.1 API-Gateway. Serverless can take over the back-end scaling of individual microservices and at the same time present a single frontend via an API gateway. The API gateway represents a single point of entry for different clients. When an API endpoint is called, intelligent routing to the back-end services takes place. The endpoints of the services can be implemented as Serverless functions so that resources are allocated automatically depending on the amount of requests [13].

2.1.2 Cronjobs. Serverless Functions can be used as cronjobs to perform regularly repetitive tasks. The trigger here is a timer that executes the Serverless function at certain time intervals. The advantage of using Serverless functions as cronjobs is that a physical or virtual machine does not have to be operated continuously. Instead, resources are allocated only at the time of execution and released again when the

task is complete. The cronjobs are also decoupled from the hosts on which a certain task is to be executed [3].

2.1.3 File Triggers & Stream Processing. Serverless functions can be used to implement classic extract, transform, and load (ETL) processes. When different events occur, code can be executed as part of a larger pipeline. For example, a function could be triggered by uploading a file. This function downloads the file while another one applies transformations to the data. Another function loads the now structured data into a database. Depending on the number of uploaded files, the ETL process scales automatically. Serverless functions can also be used when processing data streams by including appropriate logic to parse the data from the stream, bring it into a structured format, and then store it in another system. Again, the serveress function scales according to the amount of data transfered [20] [14].

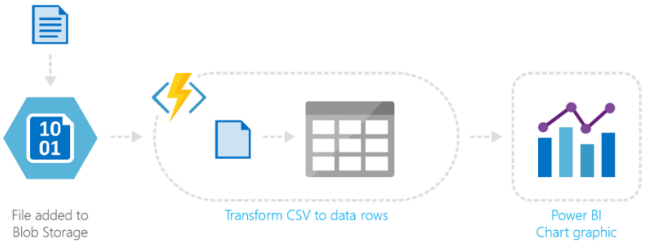

Figure 3: Example of a data pipeline implemented using a Serverless function. [16]

3 SERVERLESS SCENARIOS

This section introduces some use cases for Serverless architectures.

3.1 Mobile Backends

Mobile applications often also require server-side logic. Sometimes they need to access some server-side APIs, sometimes they need to access and work with locally stored data. In addition, these applications often need to outsource more compute-intensive tasks to a scalable cloud platform. There is typically a clear separation of responsibilities between mobile developers and back-end developers. Serverless Computing allows mobile developers to focus on developing the actual mobile application while leveraging the server-side logic that others may have implemented. This logic can be implemented as a Serverless function that can optionally be provided via an API gateway. Many cloud providers already offer runtime environments for Serverless functions in which e.g. Swift can be executed. This allows mobile developers to quickly develop simple server-side logic without having to learn another programming language. This can significantly accelerate the development of scalable mobile applications in companies without a broad programming language spectrum [12].

3.2 Internet of Things

The Internet of Things can be described as a network consisting of numerous so-called *Smart Objects* that exchange information between each other. Each intelligent object in the Internet of Things can be identified via its own internet address and can thus be addressed by humans via the world-wide-web. Although humans are responsible for controlling these intelligent objects, their influence is limited. By connecting to the internet, the smart devices are able to act independently, adapt to situations and react to certain scenarios. A big challenge in the IoT is the large number of devices that have to communicate with each other. IoT often dictates an event-controlled architecture that routes and processes messages between the individual devices. Serverless can play to its strengths here.

The use of Serverless platforms offers a highly available environment for the processing of IoT data. At the same time, Serverless functions scale with the number of IoT devices and data traffic. By deploying individual functions, it is not necessary to deploy the entire system again when changes are made, but only the individual affected function, which also makes it easier to add and support new devices and sensors without downtime [19] [23].

3.3 Chatbots

Chatbots that provide a conversational interface to interact with the user have become more and more popular in the market with wide acceptance of mobile and intelligent devices. Chatbots today are now embedded in popular messaging applications and appear as standalone services such as *Apple Siri*[1], *Amazon Alexa*[2] and *Microsoft Cortana*[3].

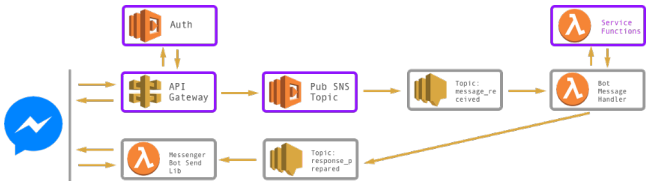

Figure 4: Exemplary architecture of a Facebook Messenger chatbot, implemented with AWS Lambda functions and topics to route messages from publishing clients to subscribing clients. [25]

[1] https://www.apple.com/de/siri
[2] https://www.amazon.de/b?ie=UTF8&node=12775495031
[3] https://www.microsoft.com/de-de/windows/cortana

Serverless Computing is an ideal platform to build extensible chatbots for various scenarios. Because of its cloud-native nature, Serverless functions can be seamlessly integrated with other cloud offerings such as artificial intelligence or natural language processing. At the same time, they take the challenge of delivering a global, scalable infrastructure away from the developer and allow him to focus on the basic functionality of the chatbot [18].

4 RESEARCH CHALLENGES

In this section, selected challenges related to Serverless Computing are explained and approaches to solving them are outlined. The challenges presented are the basis for more in-depth research.

4.1 State Management

A Serverless function, similar to a microservice, is usually stateless. A persistent state between two executions of the same function is usually not possible due to the volatile lifetime. It should be avoided to use states in order to scale the Serverless functions easily and avoid a single point of failure. However, there may well be applications in which a state is necessary. There are various ways to deal with this situation. The state can therefore neither be kept in memory nor on the hard disk, so a Serverless function must store the state externally. This can be realized e.g. by a distributed cache, the storage of the state in a database or a distributed object storage, or also by a workflow engine. However, the selected solution should scale horizontally, as they can otherwise become a bottleneck in the Serverless architecture [21] [16].

4.2 Transaction Management

Managing transactions in a Serverless architecture is a challenge. Sometimes an application needs business logic that performs multiple, "all or nothing" operations across multiple database tables and provides atomicity, consistency, isolation, and durability (ACID). Due to the short lifespan of a Serverless function, known patterns from the microservice world such as the 2-Phase commit or Saga pattern can be difficult to apply to a Serverless architecture to implement distributed transactions. One approach used is to outsource transaction management to an external system. An example of this is Amazon's *DynamoDB*[1] to implement related inserts, updates or delete operations in a single business logic operation or *Cloud Spanner*[2] as a scalable, globally distributed, and synchronously replicated database that provides transactional consistency via a SQL-like interface [22] [16].

[1]https://aws.amazon.com/de/dynamodb
[2]https://cloud.google.com/spanner

4.3 Function Startup Time

A Serverless function does not reserve any resources in advance. If the function is not executed, it will still be held by the cloud provider for a certain period of time with minimal resources, as constantly recreating the function would be time-consuming and inefficient. At the end of this period, which varies depending on the cloud provider, all resources of the function are released. If a Serverless function is triggered again, a so-called "cold start" must be performed, which can take a few seconds. The start time varies depending on the cloud provider and agreed service level agreements. However, in the end it is important to be aware of this problem as a user of Serverless services. There are different approaches to solve the problem of cold starts. For example, some cloud providers, such as AWS, offer service level agreements that guarantee that a Serverless function always remains active. Other approaches involve the implementation of keep-alive mechanisms that periodically trigger Serverless functions to keep them active. Serverless frameworks such as *OpenFaaS*[1] or *Kubeless*[2], which are based on orchestration tools such as Kubernetes and follow a containerized function approach, already have very low start times [28].

4.4 Function Execution Time

Serverless functions are designed to last from a few milliseconds to a few minutes. This also benefits cloud providers, as it becomes easier to free resources when functions are terminated and to distribute workload across multiple hosts. Often the maximum runtime of a Serverless function is limited in time by the cloud provider, thus time-consuming functions should be implemented in a different way. If a Serverless function has too long an execution time, there are several ways to deal with this situation. One may divide the function into several smaller functions. This does not change the total runtime, but the division into subtasks allows a longer execution time. This can also indicate a design error if a single Serverless function has a long execution time. A decomposition of the function can be useful in this case. Alternatively, asynchronous functions can be used. Asynchronous handling minimizes the time in which the actual process runs. So-called durable functions pause and maintaine their state while waiting for the scheduling of external processes [16].

4.5 Logging, Monitoring & Tracing

Due to the volatile nature of Serverless functions, monitoring is not entirely uncomplicated. Many cloud providers offer users their own monitoring solutions, which makes monitoring on their platform accessible. There are also many monitoring tools that are based on the monitoring solutions

[1]https://www.openfaas.com/
[2]https://kubeless.io

of the cloud providers. There are different approaches for the implementation of Serverless monitoring. Firstly, client code for a monitoring backend can be written directly to the Serverless function. Other solutions wrap the code for the Serverless function in a runtime environment that already has a client for monitoring, so that the user can keep the actual Serverless function free of third-party libraries and additional middleware. Other approaches use the cloud provider's API to access monitoring information and aggregate metrics. Nevertheless, many challenges remain open [21].

4.6 Function Failure & Resiliency

Theoretically, any service based on Serverless Functions scales automatically and also benefits from the availability and fault tolerance properties of Serverless Computing. This includes provisioning of Serverless function instances, high availability, retry logic from the cloud provider if available or seamless integration with other cloud services. However, there are still error conditions that Serverless Computing cannot cover, such as bugs in function code, unavailability of external services on which a Serverless function depends, or general code performance. In order to prevent this from happening, the tried and tested principles of error handling can be applied to the Serverless function. For example in case of basic HTTP functions, the right approach is for the code that calls the function to perform the retries. To implement retries, developers can either write their own retry logic or reuse a common library . Other approaches, as mentioned above, outsource the retry logic to the cloud provider. Here the execution of the Serverless function is automatically retried at a certain received event until it is finally completed successfully. Backoff policies can be used to configure at which intervals the retries are to be executed [27].

4.7 Debugging & Testing

Individual Serverless functions are usually quite small and manageable. The actual complexity of the application results from its composition. Unit testing should therefore prevent a function with programming errors from going into production. A local detailed analysis with profilers and debuggers is possible, but difficult for functions running in production on the Serverless platform. Integration tests, on the other hand, are much more difficult to implement in Serverless applications than unit tests. Appropriate integration test environments can be set up on the Serverless platform alongside the productive environment. However, it should be considered that the tests use resources in the cloud that have to be paid for accordingly. Regular load tests are therefore not necessarily an option. Depending on the cloud provider, test instances for required services or even stubs for mocking interfaces of the Serverless platform can be provided, but no local environment can replace the cloud environment in which the Serverless application ultimately runs [21] [2].

4.8 Vendor Control / Lock-In

A much debated issue in Serverless Computing is the risk of vendor lock-in, i.e. being tied to vendor-specific interfaces or services. The possibility of vendor lock-ins is often the reason why Serverless Computing is not chosen as the paradigm for application development. The reasons for independence from a cloud provider can be very different, e.g. a user of a Serverless platform does not want to enter into a long relationship with the provider and only use services if they provide advantages for his own application. Also the idea that one's own application is dependent on the further development of the underlying cloud platform is also often a deterrent. Nevertheless, a lock-in does not have to be negative. A user of a Serverless platform has to consciously get involved and consider the possible consequences. Currently, there are many efforts to ensure vendor independence in Serverless Computing. One example is the abstraction framework with the name *Serverless*[1]. It was originally designed to simplify development in the AWS cloud, but now also supports the other three major players (Microsoft Azure Functions, Google Cloud Functions and IBM OpenWhisk) within an acceptable scope. Another framework is the *Spring Cloud Function*[2] project, which provides a unified Function-as-a-Service programming model across a variety of cloud providers [6] [15].

4.9 Tooling

Due to the still young Serverless Computing paradigm, the availability of tools for the deployment, compilation and configuration of Serverless applications is not yet very developed. Although frameworks like the Serverless Framework take a lot of work out of it, tools and IDEs for Serverless Computing are missing that integrate into the developer workflow and improve the user experience of Serverless developers. There is also a need for tools for monitoring Serverless functions and remote debugging [21].

The challenges presented are only an outline of many others. Other aspects related to Serverless Computing which can serve as a basis for in-depth research are e.g. versioning, security aspects, multi-tenant capability [...] of Serverless functions.

[1] https://serverless.com

[2] https://cloud.spring.io/spring-cloud-function

5 OPPORTUNITIES

This section provides an overview of the opportunities Serverless Computing offers.

5.1 Development Focus & Prototyping

Through interactive prototypes, abstract concepts become concrete and above all can be experienced and experienced by end-users. In this way, the scope for interpretation for users is kept to a minimum. Even for the developers, it quickly becomes clear without long documentation how something can be implemented. In this way, all content and functions relevant to the user can be worked out before an unnecessary amount of money and time is invested in numerous implementation phases. Serverless Computing is about leveraging tighter integration to cloud provider services. Creating Serverless applications means that developers can focus on the product instead of worrying about managing and running servers or runtimes in the cloud or locally. This reduced effort can save developers time and effort that can be used to develop products that are scalable and reliable. This leads to quick prototyping of ideas and reduces development time. Using Serverless, a Minimum Viable Product can be scaled and used as a final product if built correctly and with required changes. This is especially interesting for startups to save costs and time.

5.2 Billing Model

A particular characteristic of Serverless Computing is the way in which the consumed resources are charged. The operation of own hardware is associated with a variety of costs, e.g. for maintenance, premises or staff. Cloud computing, on the other hand, makes it possible to rent computing power from the cloud provider, which is why a large part of the maintenance costs for the infrastructure is eliminated. However, idle times of the rented infrastructure are still calculated. Serverless Computing goes one step ahead and promises to reduce infrastructure costs even further. A Serverless function is volatile, the allocated resources are released again at the end of execution and only the actual execution period is charged. The expenses for idle times are eliminated and only the actually used computing power is paid for. Serverless also leads to reduced operating costs, as more work is the responsibility of the cloud provider [8].

5.3 Polyglot Programming

Similar to Microservices, Serverless functions allow the free choice of programming languages. Depending on the application, developers can choose between different language runtimes. These functions are loosely linked via events. Each Serverless provider offers a range of runtimes, which can differ from provider to provider. In general, all major programming languages are supported, such as Java, Python, and Node.js [11].

5.4 Autoscaling

Serverless enables automatic scaling of the application without having to worry about configuration. With classic services, it must be decided either manually or on the basis of metrics whether more instances must be started as the load increases. With Serverless Computing, the cloud provider takes over this responsibility and can increase function instances depending on the load. However, it must be noted that by giving control of scaling to the cloud provider, it is difficult not to be able to address and mitigate errors related to new Serverless instances.

6 CONCLUSION

The Serverless applications are promised a rosy future by various voices. The term Serverless Computing refers to a consistent further development of hybrid cloud structures and microservices. It takes the burden off the developers to deal with infrastructure and concentrate on the development and behavior of their applications. Serverless accelerates the development process, simplifies operations management, and reduces development and operational costs. The simplicity that this concept brings to software development will make it mature and establish faster than any other paradigm. However, critics of Serverless Computing warn that Serverless users are increasingly losing control of their projects. Due to the early development phase at the time of this work, there are still many deficits such as strategies for integration testing, monitoring and debugging tools, which is why despite the significant advantages of Serverless Computing, attention should be paid when deciding to use this paradigm.

In summary, there is at this time a lot of development in different aspects in the area of Serverless Computing, which is why it remains exciting how the term Serverless Computing and the Serverless platforms will develop in the future.

REFERENCES

[1] Amazon. 2014. Introducing AWS Lambda. https://aws.amazon.com/about-aws/whats-new/2014/11/13/introducing-aws-lambda/ Accessed: 2019-2-14.

[2] Amazon. 2018. Testing and Debugging Serverless Applications. https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-test-and-debug.html Accessed: 2019-1-23.

[3] Amazon. 2019. Schedule Expressions Using Rate or Cron. https://docs.aws.amazon.com/lambda/latest/dg/tutorial-scheduled-events-schedule-expressions.html Accessed: 2019-2-25.

[4] Abel Avram. 2016. FaaS, PaaS, and the Benefits of the Serverless Architecture. https://www.infoq.com/news/2016/06/faas-serverless-architecture Accessed: 2019-2-23.

[5] Christian Bargmann. 2018. Serverless & FaaS. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2018-gsem/Bargmann/folien.pdf Accessed: 2019-1-11.

[6] Philipp Beyerlein. 2018. Serverless - The Vendor Is Not Your Enemy. https://www.innoq.com/de/blog/serverless-vendor-is-not-your-enemy/ Accessed: 2019-2-15.

[7] Cloudflare. 2018. How are serverless computing and Platform-as-a-Service different? https://www.cloudflare.com/learning/serverless/glossary/serverless-vs-paas/ Accessed: 2019-2-14.

[8] Cloudflare. 2018. What Is Function as a Service (FaaS)? https://www.cloudflare.com/learning/serverless/glossary/function-as-a-service-faas/ Accessed: 2019-2-14.

[9] A. Eivy. 2017. Be Wary of the Economics of "Serverless" Cloud Computing. *IEEE Cloud Computing* 4, 2 (March 2017), 6–12. https://doi.org/10.1109/MCC.2017.32

[10] Geoffrey Fox, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2017. Status of Serverless Computing and Function-as-a-Service(FaaS) in Industry and Research. Unpublished. https://doi.org/10.13140/RG.2.2.15007.87206

[11] Shekhar Gulati. 2018. hands-on-serverless-guide. https://github.com/shekhargulati/hands-on-serverless-guide/blob/master/01-aws-lambda-serverless-framework/01-introduction-to-serverless.md Accessed: 2019-2-23.

[12] Adrian Hall. 2018. How to design a serverless backend that scales with your apps success. https://read.acloud.guru/build-a-cost-effective-mobile-backend-for-scale-and-security-4c0c143e898c Accessed: 2019-1-25.

[13] HashiCorp. 2018. Serverless Applications with AWS Lambda and API Gateway. https://learn.hashicorp.com/terraform/aws/lambda-api-gateway Accessed: 2019-1-17.

[14] Sanghyun Hong, Abhinav Srivastava, William Shambrook, and Tudor Dumitras. 2018. Go Serverless: Securing Cloud via Serverless Design Patterns. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/hotcloud18/presentation/hong

[15] Serverless Inc. [n. d.]. The most widely-adopted toolkit for building serverless applications. https://serverless.com Accessed: 2019-2-26.

[16] Cecil Phillip Jeremy Likness. 2018. *Serverless apps: Architecture, patterns and Azure implementation.*

[17] Ali Kanso and Alaa Youssef. 2017. Serverless: Beyond the Cloud. In *Proceedings of the 2Nd International Workshop on Serverless Computing (WoSC '17)*. ACM, New York, NY, USA, 6–10. https://doi.org/10.1145/3154847.3154854

[18] Jyri Lehvä, Niko Mäkitalo, and Tommi Mikkonen. 2018. *Case Study: Building a Serverless Messenger Chatbot.* 75–86. https://doi.org/10.1007/978-3-319-74433-9_6

[19] Duarte Pinto, João Pedro Dias, and Hugo Sereno Ferreira. 2018. Dynamic Allocation of Serverless Functions in IoT Environments. *CoRR* abs/1807.03755 (2018). arXiv:1807.03755 http://arxiv.org/abs/1807.03755

[20] Danilo Poccia. 2018. Building Serverless ETL Pipelines. https://de.slideshare.net/AmazonWebServices/building-serverless-etl-pipelines Accessed: 2019-1-10.

[21] Mike Roberts. 2018. Serverless Architectures. https://martinfowler.com/articles/serverless.html Accessed: 2019-2-12.

[22] Bernd Rücker. 2019. Lost in transaction? Strategies to manage consistency in serverless architectures. https://berndruecker.io/lost-in-transaction/ Accessed: 2019-2-16.

[23] Raman Sharma. 2018. Go serverless for your IoT needs. https://azure.microsoft.com/de-de/blog/go-serverless-for-your-iot-needs/ Accessed: 2019-1-24.

[24] Matt Stine. 2017. Serverless? Not so FaaS! https://www.youtube.com/watch?v=e59pTjaVPCs Accessed: 2019-1-10.

[25] Mustafa Turan. 2017. A Serverless, Event Driven Architecture for Chatbots. https://chatbotsmagazine.com/a-serverless-event-driven-architecture-for-chatbots-3095eb40cbb7 Accessed: 2019-1-23.

[26] Tim Wagner. 2016. Getting Started with AWS Lambda and the Serverless Cloud. https://de.slideshare.net/AmazonWebServices/getting-started-with-aws-lambda-and-the-serverless-cloud/29%20(Accessed: 2019-2-26.

[27] Slawomir Walkowski. 2018. Cloud Functions pro tips: Using retries to build reliable serverless systems. https://cloud.google.com/blog/products/serverless/cloud-functions-pro-tips-using-retries-to-build-reliable-serverless-systems Accessed: 2019-2-15.

[28] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael M. Swift. 2018. Peeking Behind the Curtains of Serverless Platforms. In *USENIX Annual Technical Conference*.