Security Implications of Serverless Technologies

Research Proposal

Tom Hinton

Contents

1. Introduction 3
1.1 Benefits of Serverless 3
1.2 Drawbacks of Serverless 4
2. Background to the Proposed Research 6
3. Aims & Objectives 6
4. Research Methods 8
4.1 Philosophy 8
4.2 Strategy 8
4.3 Approaches 8
4.3.1 Experiment 8
4.3.2 Document Review 8
4.3.3 Software Development 8
4.4 Risks & Limitations 9
4.5 Data Evaluation 9
5. Development Methods 11
5.1 Software Development Methodology 11
6. Resources & Planning 12
6.1 Resources Required 12
6.2 Gantt Chart 12
7. Reference List 13
8. Bibliography 15

1. Introduction

This project will focus on 'Function-as-a-service' (FaaS), which is a stateless system i.e. one which executes logic but which does not carry persistent information. This is a part of a new generation of software development known as 'Serverless'. This moniker is potentially misleading. It would be reasonable for a layperson to expect that 'Serverless' technology refers to applications which run without the use of servers. This is not, however, the case. Serverless applications do in fact run on servers but not local to the user, instead user-built functions are provisioned and run from servers which are managed by the service provider. The advantage of this is that the user does not need to anticipate their application's hardware load requirements. Instead, space on cloud-based servers is allocated based on demand which allows the user to scale up and down depending on their needs which can result in a significant saving in costs. [1]

With Serverless, the developer writes functions which are fired by 'triggers', which can be defined as some form of event that can cause a Lambda function to run. Some trigger types, or 'sources', invoke the function directly. Other sources place the data into a queue and invoke Lambda functions at the top of the queue. This arrangement means that cloud-based storage can be used with the same functionality as a locally-run application, since any data stored within the Lambda function itself is volatile and may not be retrievable once the function completes.

As a result of the above, developers tend to use a more 'Microservice-oriented' approach, i.e. a large number of small modules that cooperate to produce a solution, as opposed to the 'Monolithic approach' where a single, highly-coupled entity is able to independently service all requests. [2] Serverless development necessarily takes place within 'stateless' containers, which refers to the property of all memory of the container being volatile. The containers are instantiated to meet demand and then destroyed when no longer required. As a result of this, it is not wise to assume that data persists natively between runs of a function. [3]

1.1 Benefits of Serverless

One aspect of Serverless which is generally extremely well-received by developers is the fact that it is not necessary to engage in server maintenance and support. Managing and maintaining a large set of servers is normally an onerous task. An example of this would be the tool that was created by Whitaker, Cox and Gribble [4] to detect failures in server configurations. Another example of this would be the hill-climbing algorithm written by Xi et al [5] for the same purpose.

A cost-saving can occur while using Serverless applications as they charge the owner only when they are in use. As they are not chargeable when they are idle, this can significantly cut down on costs. [6] Furthermore, the costs generally associated with the setup and maintenance of physical servers, for example rack space, are also avoided. Projects being undertaken by small companies or companies which do not choose to make significant capital investment in new projects are therefore well-suited to this environment.

The phenomenon known as 'scaling up', whereby applications automatically create new instances of themselves to service greater numbers of requests, is inherently more elastic in its nature. [7]

'Scaling down', which means that instances are terminated in times of low traffic to save on costs, means that greater efficiency is achieved. This is however not without its disadvantages discussed below.

A major advantage of Serverless is its high availability. Architecture which uses a network of geographically separated servers is often employed by cloud service providers. This means that Serverless is less vulnerable to natural disasters, power outages and even politically motivated hostile attacks. AWS's Lambda service is an excellent example of this. It runs on multiple data centres which are split into Availability Zones depending upon the geographical area in which the centres are located. [8] A similar approach is adopted by Microsoft Azure. [9]

1.2 Drawbacks of Serverless

The rise in popularity of Serverless applications has been rapid but there are a number of issues which developers should be aware of before deciding upon using a cloud provider for their project.

It is the provider that has partial responsibility for a potentially critical proportion of the implementation and infrastructure of Serverless applications. Vendor-exclusive APIs and features are therefore at risk of becoming coupled to the development of a Serverless application. This becomes a problem if a need to change providers is identified. [10]

Availability zones mean that Serverless applications are robust with regard to security. Nevertheless, it must be borne in mind that all sensitive data in the application is being handled by the cloud provider. While developers may not see this as a cause for concern with some projects there may be cases where this is an unacceptable breach of security. Furthermore, developers have little control over the way that their application is run due to the large amount of automation inherent in Serverless applications. This leads to a unique security environment that has, at time of writing, not yet been completely explored.

In terms of cost, Serverless applications are generally accepted to be cheaper than maintaining a server which is local to the developer. [6] This, however, may not necessarily be the case as the costs involved may be very variable. If periods of high traffic occur without warning, this will incur costs which will inevitably need to be met. This has also led to a new form of malicious attack known as 'Denial of Wallet' (sec ref) whereby a Serverless application is forced to respond to an overload of dummy requests which results in heavy expenditure. These unexpected costs by their nature cannot be foreseen and are therefore a threat to traditional budget management.

Serverless is a new technology and while this is not an inherent weakness, it means that it requires a learning curve on the part of the developer which may not be as well-supported as more conventional technologies. Learning how to develop with an architecture using functions will inevitably take time and therefore costs. The inchoate nature of the field is where the justification of this project stems from.

The first request made to an inactive Serverless application causes it to scale up and therefore create an instance to service the request. This property affords resource efficiency in times of inactivity but comes with an inherent flaw. There is a noticeable time delay while this instance is provisioned, which can result in an initially sluggish response time, sometimes in the order of seconds. This problem is generally known as the 'cold start' problem. [2] While this may not seem significant, it may impact on processes running on other machines and can lead to a cascade into other cold starts, possibly resulting in failure. The relationship between cold start times and a variety of influencing factors is discussed by Cui. [11]

The cold start times issue is mainly a problem for the cloud service provider as it is related directly to the efficiency of their hardware. [12] The developer can, however, mitigate the risks of a cold start issue by not using statically-typed languages. Cui shows that this is known to decrease response times. [11] Another useful technique is called 'pre-warming', where a timed function is used to

stimulate the application and thus to keep a container active. There are cost implications for this but they may be considered tolerable if it is necessary for response times to be minimised.

2. Background to the Proposed Research

This project focuses on Amazon's cloud service which is called Amazon Web Services (or AWS). [13]

Amazon started offering cloud computing services in 2006 with two services – the first was a storage service that allows clients to send data to online repositories called buckets and was known as Simple Storage Service or S3. The second was the cloud computing service that gives clients the ability to create virtual machines using Amazon's hardware and run them from the cloud. This service is called Elastic Compute Cloud or EC2.

In 2015, AWS Lambda was launched. [14] This is the service that allows users to create 'Lambda functions'. In the context of cloud computing, a Lambda function is defined as follows: "The code you run on AWS Lambda is uploaded as a 'Lambda function'. Each function has associated configuration information, such as its name, description, entry point, and resource requirements." [15]Since its inception, Lambda has gained additional functionality including function authorship in Java and Python and tie-ins with Amazon Lex, the chat bot service.

AWS has become a powerful tool for developers that choose to create Serverless applications but as a new field it is important to keep track of how to maintain the security of these systems. Many types of attack on applications are no longer a concern when using Serverless infrastructure thanks to a combination of the stateless nature of containers and Amazon's vested interest in protecting their own hardware from being broken into. However, the security of Lambda functions and Serverless applications is a shared responsibility between AWS and the developer. [16]

Security companies are starting to take notice of these vulnerabilities and offer recommendations and services to help combat them. [17] This is the focus of this project: to identify, demonstrate and record exploits in AWS Lambda that may not be well-known thanks to the relative newness of the platform. As the majority of the information on vulnerabilities will mostly likely be taken from other security companies, this work will have significant overlap and serves to provide a reference for how to recognise weaknesses that have already been identified by others.

It is anticipated that this project will produce a dummy Serverless application with demonstrable vulnerabilities and a compilation of those vulnerabilities. This may need to change depending on time constraints for each section.

3. Aims & Objectives

The aim of this project is as follows: to look into the structure of a practical Serverless-based system with particular emphasis on security vulnerabilities.

In order to fulfil this aim, the following objectives will need to be met:

- Research some existing vulnerabilities in Serverless systems

  This objective is tied to the literature review. It is an information-gathering objective which will inform the direction of the rest of the project. This involves discovering what properties the dummy system will need in order to be a representative Serverless system, then choosing AWS components to replicate that functionality. This objective also contains a

decision point as to whether to build the dummy system or not, depending on time constraints.

- Design and build a Serverless system with the intent of compromising it

  This objective handles the creative aspect of the project. It involves creating a design for the example Serverless system then investigating potential security exploits. These vulnerabilities would then be placed into the dummy system's design. At this point depending on the events described in the objective above, the design would be built. This would involve taking care not to secure against any exploits (as demonstrating their function is the driving force of the project).

  This is also the objective that handles the search for tooling and scripts to activate the dummy system and break into it. Terraform is currently the most likely programmatic solution but this may change as the literature review progresses.

- Demonstrate potential exploits of Serverless systems and how they may be recognised and prevented

  This is a presentation objective that handles the marshalling and presentation of the work completed previously in the project. A demonstration of the identified vulnerabilities will be required and will be made at this point. This is also the point where the sources used in the literature review are compiled and the content of the report written. Possibly also at this point a poster may be made to meet the brief of the project.

4. Research Methods

4.1 Philosophy

This project will use a positivistic philosophy – as the brief is to examine and provide recommendations on security measures for AWS, the project will be focussed on observation of real world entities, namely Serverless applications and exploits. Phenomenology would be an inappropriate way to answer the research question as the knowledge that is desired is objective. The desired information is specifically based around ways to break into Serverless systems and therefore is measurable.

The newness of the field is another factor in selecting a positivistic approach as there is less material to analyse and survey than there is for more established topics. Serverless is still in its infancy which is part of the justification for this study.

4.2 Strategy

The approach for this project will be inductive. While this is not the norm for positivistic projects, this project's aim is to discover the best ways to cause and guard against security issues in Serverless architecture. Consequently, the hypothesis is not yet known. Instead, this project aims to create its hypothesis after research has been done into Serverless security which means the hypothesis cannot drive the direction of the work.

4.3 Approaches

4.3.1 Experiment

The experiment will be the business value derived from the completion of the software build. The dummy application itself has no function but the ways that it can be broken into will be useful to know and this is where this section becomes relevant.

Each exploit that is found during the document review will be used against the dummy system. The logging that is built into the system will be used to measure how successful each attack is and what systems are affected.

This section of the project is arguably the most valuable as it forms the backbone of the presentation material however it is possible that a dummy system is never created and a replacement is never found. In this scenario the final deliverables will emphasise discoveries with regards to existing security material.

4.3.2 Document Review

The document review is critical to most projects and this one is no exception. Data on Serverless will be compiled so as to find vulnerabilities in it and ways to protect an application from these vulnerabilities. It may also inform the direction of the project to find out whether this question has already been answered, hence why it is necessary.

4.3.3 Software Development

In order to fully understand how an exploit can be used to break into a Serverless system, a simple Serverless dummy system will need to be made in order to attack it. Additionally, tools and scripts will be needed to set up and run these tests promptly to avoid being charged unnecessarily (account to be provided by sponsor). Terraform is most likely to be used to serve this purpose.

Before the system is made, research will be done to determine which properties the dummy system should have in order to represent a practical, real-world system. Once these properties are known, research will be done into how to represent these components using Serverless functions.

The plan makes allowance for time constraints – it is unknown whether this system will be buildable or not. Whatever path the project takes however, it will always need to make a design for a system that could be used as a test bed for exploits.

The system will need probes and loggers added to it, if practicable so that it is possible to search for changes after an exploit is attempted.

4.4 Risks & Limitations

There is a danger in this project that the dummy system that is being analysed does not have the correct features to be representative of an average Serverless application. This would be due to insufficient data or an incorrect interpretation of it during the design stage. The consequences of this would be that the relevance of the security exploits found in the project may be questionable but the report would still have value for its study of the topic of Serverless.

This project is motivated by the interests of the author's sponsor. The sponsor is currently interested in expanding into Serverless architecture, specifically using Amazon Web Services (AWS), hence this project's focus on AWS.

The research is innately coupled to the service provider, as different services such as Microsoft Azure may have different underlying structures and therefore different weaknesses. If the sponsor were to decide to switch platform or strategy, it would be more difficult to accomplish the tasks necessary for the project.

There are certain attacks that rely on taxing the host's resources to cause a fault. It would be best, academically, to run these attacks against my weakened system especially because in Serverless the attack can have different effects: for an instance-limited application it causes a Denial of Service attack but for an unlimited one the massive creation of containers causes a Denial of Wallet attack.

However, studying this would be impracticable. Undeclared attacks against Serverless applications are detected by Amazon and automatically shut down – a useful countermeasure but problematic for testing. Denial of Wallet attacks can also be very expensive to demonstrate, causing financial loss for either the individual or the sponsor.

There is a weakness within the project in that it is predicated on exploiting vulnerabilities of a dummy system. Fundamentally, this is not a real system and therefore cannot catch errors that arise due to complicated interactions between multiple moving parts.

4.5 Data Evaluation

The research into the test system will contain some kind of analysis of the commonality of functionality and components in a Serverless system. This data is critical as it informs the nature of the dummy application.

The most important data to process will be the tests to measure changes of state in the dummy system after being attacked by various security exploits. Data will be compared before the attempt was made to attack the system but also to the expected outputs of the attacks – this will help inform whether an attack has succeeded but also whether an attack has unknown consequences. The data

itself does not require statistical analysis but interpreting why certain changes may have been made after certain attacks will be important for the project.

Specifically, care must be taken to identify whether an exploit is successful. If it is, it must be determined whether a change is deliberate and due to an exploit or accidental and associated with some other error. Probes and logs built into the Serverless system should aid in detecting these errors.

5. Development Methods

5.1 Software Development Methodology

The software development of the project is the dummy system for testing exploits against, provided that time allows and also any scripts that will be needed to initiate the tests. The system will be build using a Kanban structure that guides the rest of the project as a whole.

The direction of the project will be informed by fortnightly meetings with the supervisor. This will help maintain a realistic scope for the project.

Both the methodology and tasks have been designed to be as flexible as possible. Serverless is a relatively new technology and consequently has a small community of developers and researchers which can be used as a resource and support network. There is a danger that this task's complexity ranges beyond the level of this study and that the relative inexperience of the researcher cannot be mitigated by the experience of the research community because, as previously discussed, that community is at this point, still quite small. For this reason, it may also be difficult to scope accurately. The final stage of the analysis phase therefore includes a decision point which is the point at which consideration must be given as to whether there is time to build the test project or merely define it for a future work.

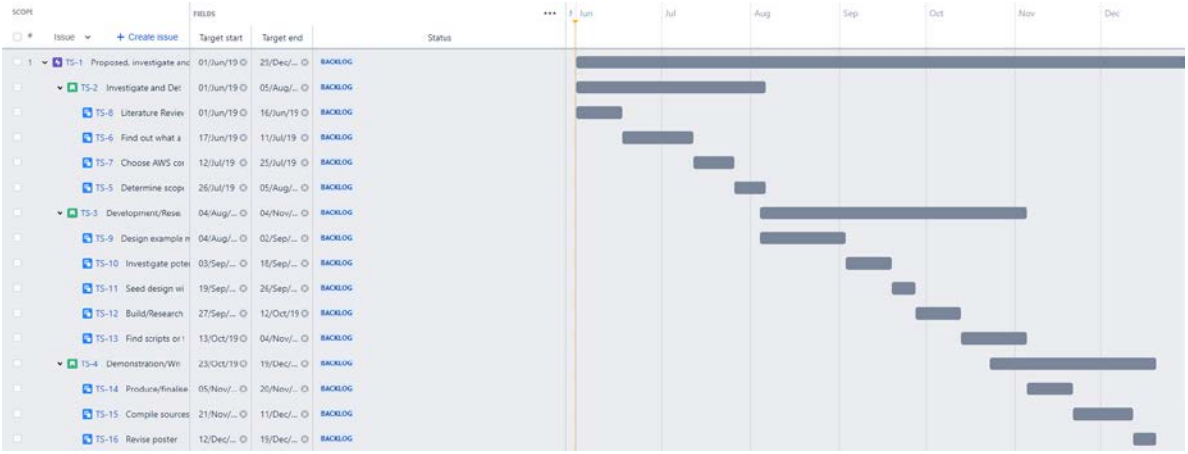
6. Resources & Planning

6.1 Resources Required

An AWS account with privileges to create the system to attack and then to attack it. A package such as Terraform will also be required to be able to boot and destroy resources rapidly.

A development machine that can handle snippets of Java code and an Integrated Development Environment to develop these. This allows development of individual functions in an environment where they can be tested rigorously.

6.2 Gantt Chart



7. Reference List

[1] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau, “Serverless Computation with OpenLambda,” in *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '16)*, Denver, 2016.

[2] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian et al, “Serverless Computing: Current Trends and Open Problems,” *Research Advances in Cloud Computing,* pp. 1-20, 2017.

[3] E. van Eyk, A. Iosup, S. Seif and M. Thömmes, “The SPEC Cloud Group's Research Vision on FaaS and Serverless Architectures,” in *First International Workshop on Serverless Computing (WoSC)*, Las Vegas, 2017.

[4] A. Whitaker, R. S. Cox and S. D. Gribble, “Configuration Debugging as Search: Finding the Needle in the Haystack,” in *6th Symposium on Operating Systems Design and Implementation (OSDI)*, San Francisco, 2004.

[5] B. Xi, Z. Liu, M. Raghavachari, C. H. Xia and L. Zhang, “A Smart Hill-Climbing Algorithm for Application Server Configuration,” in *World Wide Web Conference*, New York, 2004.

[6] M. Villamizar, O. Garcés, L. Ochoa, H. Castro et al, “Infrastructure Cost Comparison of Running Web Applications in the Cloud using AWS Lambda and Monolithic and Microservice Architectures,” in *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Cartagena, 2016.

[7] A. Pérez, G. Moltó, M. Caballer and A. Calatrava, “Serverless Computing for Container-based Architectures,” *Future Generation Computer Systems,* vol. 83, pp. 50-59, 2018.

[8] Amazon Web Services, Inc., “Regions and Availability Zones - Amazon Elastic Compute Cloud,” 10 April 2019. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html.

[9] Microsoft Corporation (MS), “What are Azure Availability Zones?,” 10 April 2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/availability-zones/az-overview.

[10] S. K. Mohanty, G. Premsankar and M. Di Francesco, “An evaluation of open source serverless computing frameworks,” in *IEEE 10th International Conference on Cloud Computing Technology and Science (CloudCom)*, Nicosia, 2018.

[11] A Cloud Guru, “Y. Cui,” A Cloud Guru, 15 June 2017. [Online]. Available: https://read.acloud.guru/does-coding-language-memory-or-package-size-affect-cold-starts-of-aws-lambda-a15e26d12c76 . [Accessed 10 April 2019].

[12] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly and S. Pallickara, “Serverless Computing: An Investigation of Factors Influencing Microservice Performance,” in *IEEE International Conference on Cloud Engineering (IC2E)*, Orlando, 2018.

[13] Amazon Web Services, “About AWS,” Amazon, [Online]. Available: https://aws.amazon.com/about-aws/. [Accessed 31 May 2019].

[14] Amazon Web Services, “AWS Lambda Releases - AWS Lambda,” Amazon, [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/history.html. [Accessed 31 May 2019].

[15] Amazon Web Services, “AWS Lambda - FAQs,” Amazon, [Online]. Available: https://aws.amazon.com/lambda/faqs/. [Accessed 31 May 2019].

[16] Amazon Web Services, “Security Overview of AWS Lambda,” Amazon, March 2019. [Online]. Available: https://d1.awsstatic.com/whitepapers/Overview-AWS-Lambda-Security.pdf. [Accessed 31 May 2019].

[17] Protego, “OWASP Serverless Security Top 10 | Protego,” Protego, [Online]. Available: https://www.protego.io/guide/owasp-serverless-top-10/. [Accessed 31 May 2019].

8. Bibliography

[1] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau, "Serverless Computation with OpenLambda," in *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '16)*, Denver, 2016.

[2] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian et al, "Serverless Computing: Current Trends and Open Problems," *Research Advances in Cloud Computing,* pp. 1-20, 2017.

[3] E. van Eyk, A. Iosup, S. Seif and M. Thömmes, "The SPEC Cloud Group's Research Vision on FaaS and Serverless Architectures," in *First International Workshop on Serverless Computing (WoSC)*, Las Vegas, 2017.

[4] A. Whitaker, R. S. Cox and S. D. Gribble, "Configuration Debugging as Search: Finding the Needle in the Haystack," in *6th Symposium on Operating Systems Design and Implementation (OSDI)*, San Francisco, 2004.

[5] B. Xi, Z. Liu, M. Raghavachari, C. H. Xia and L. Zhang, "A Smart Hill-Climbing Algorithm for Application Server Configuration," in *World Wide Web Conference*, New York, 2004.

[6] M. Villamizar, O. Garcés, L. Ochoa, H. Castro et al, "Infrastructure Cost Comparison of Running Web Applications in the Cloud using AWS Lambda and Monolithic and Microservice Architectures," in *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Cartagena, 2016.

[7] A. Pérez, G. Moltó, M. Caballer and A. Calatrava, "Serverless Computing for Container-based Architectures," *Future Generation Computer Systems,* vol. 83, pp. 50-59, 2018.

[8] Amazon Web Services, Inc., "Regions and Availability Zones - Amazon Elastic Compute Cloud," 10 April 2019. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html.

[9] Microsoft Corporation (MS), "What are Azure Availability Zones?," 10 April 2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/availability-zones/az-overview.

[10] S. K. Mohanty, G. Premsankar and M. Di Francesco, "An evaluation of open source serverless computing frameworks," in *IEEE 10th International Conference on Cloud Computing Technology and Science (CloudCom)*, Nicosia, 2018.

[11] A Cloud Guru, "Y. Cui," A Cloud Guru, 15 June 2017. [Online]. Available: https://read.acloud.guru/does-coding-language-memory-or-package-size-affect-cold-starts-of-aws-lambda-a15e26d12c76 . [Accessed 10 April 2019].

[12] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly and S. Pallickara, "Serverless Computing: An Investigation of Factors Influencing Microservice Performance," in *IEEE International Conference on Cloud Engineering (IC2E)*, Orlando, 2018.

[13] Amazon Web Services, "About AWS," Amazon, [Online]. Available: https://aws.amazon.com/about-aws/. [Accessed 31 May 2019].

[14] Amazon Web Services, “AWS Lambda Releases - AWS Lambda,” Amazon, [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/history.html. [Accessed 31 May 2019].

[15] Amazon Web Services, “AWS Lambda - FAQs,” Amazon, [Online]. Available: https://aws.amazon.com/lambda/faqs/. [Accessed 31 May 2019].

[16] Amazon Web Services, “Security Overview of AWS Lambda,” Amazon, March 2019. [Online]. Available: https://d1.awsstatic.com/whitepapers/Overview-AWS-Lambda-Security.pdf. [Accessed 31 May 2019].

[17] Protego, “OWASP Serverless Security Top 10 | Protego,” Protego, [Online]. Available: https://www.protego.io/guide/owasp-serverless-top-10/. [Accessed 31 May 2019].

[18] J. Chapin and M. Roberts, What is Serverless?, Sebastopol: O'Reilly Media, 2017.

[19] G. Adzic and R. Chatley, “Serverless Computing: Economic and Architectural Impact,” in *European Software Engineering Conference ESEC/FSE 2017*, Paderborn, 2017.