Technical Report 3 - DevOps

Team ThisIsUnSAFe:
Richard How (28742826), Quang Nghiep Ly (28688856), Giahuy Truong (27790673), Sriram Viswanathan (29175755), Zexian Wu (28771923)

Due date:
16 November 2020

TABLE OF CONTENTS

1. TABLE OF ACRONYMS

Acronym	Term
AWS	Amazon Web Services
BaaS	Backend as a Service
FaaS	Function as a Service
GCP	Google Cloud Platform
MS	Microsoft
IaaS	Infrastructure as a Service
IAM	Identity Access Management
IBM	International Business Machines Corporation
PaaS	Platform as a Service
SPMD	Student Project Management Dashboard
HTTP	Hypertext Transfer Protocol
API	Application Transfer Protocol

2. INTRODUCTION

It is not uncommon to often come across new buzz words in the world of IT. These refer to new technologies that disrupt the process of software development and capabilities [1]. One such technology which has been recently gaining a lot of traction is Serverless Architecture. Our report sets out to explore this architecture and evaluates whether it would be appropriate as future Monash units learning materials.

3. SERVERLESS ARCHITECTURE

Serverless Architecture was initially introduced in the year 2014 by one of the world's leading Cloud Service Providers, Amazon Web Services, with the introduction of their popular AWS Lambda service. The use for this technology has only catapulted from there with more cloud platforms adopting this in their suite of services such as Microsoft Azure, Google Cloud Platform, etc.

A common saying for serverless technology is to "focus on your application, not the infrastructure" [2]. This is because Serverless Architecture focuses on relegating the infrastructure management tasks to third-party cloud services so that developers can solely focus on developing products that provide business value. With less administrative overheads, building more agile and change-responsive applications is possible [3]. Some of the renowned companies that use the Serverless Architecture are Netflix, Reuters, AOL, etc. For example, Netflix uses the AWS Lambda service for streaming content for its customers without the need for running and managing servers [4].

Some of the current popular serverless cloud services include:

- AWS Lambda - AWS
- Google Cloud Functions - GCP
- Azure Functions – Microsoft Azure
- IBM OpenWhisk – IBM Cloud
- Auth0 Webtask – Auth0

Although the term serverless may indicate that there are no servers involved, it is to be noted that this is not the case. In fact, physical servers are indeed necessary, but developers do not need to be concerned about how these servers will be provisioned and maintained.

There are 2 types of serverless computing, with some overlap: BaaS and FaaS [5].

BaaS is shorthand for "Backend as a Service" which refers to applications that use a vast majority of third-party cloud services to handle server-side logic such as authentication services, cloud storage, database management and more.

On the other hand, FaaS, which stands for “Functions as a Service” refers to the server-side logic that is run in stateless compute containers. These are event-driven and ephemeral in nature which is fully managed by a third-party cloud service. It enables developers to easily build, run and maintain their applications in the form of functions without having to manage their own infrastructure [6].

Although these two overlap in certain aspects, serverless computing, at its core, relates more to FaaS [7].

4. COMPARISON OF ALTERNATIVES

4.1 Serverless vs Microservices

By using Microservices Architecture, the application is broken into smaller components that are independent of each other. Microservices are separated and work together to accomplish the same tasks, rather than the traditional approach of building everything as a single component. Therefore, changes to a component would be isolated and not affect the entire application. This architecture emphasizes granularity, light weight, and reusability as similar processes are shared among multiple applications [8]-[10].

Serverless Architecture and Microservice Architecture share some characteristics, but there are some differences between them. Table I provides a brief comparison between Serverless and Microservices architecture.

Table I: Difference between Microservice and Serverless Architectures

Characteristics	Microservice Architecture	Serverless Architecture
Granularity	Microservice	Nanoservice/ function (A single microservice consists of several nanoservices)
Communication	HTTP API	Event trigerring
Environment	Spin up containers	Use server provided by a third-party cloud provider
Scalability	Scales differently to each microservice	Scales differently to each function
Testing	Lower granularity so easier to test; well supported by tools and processes	Higher granularity; harder to debug, troubleshoot and test
Time to Market	Design required before applying new features	Easier to design, and deploy
Cost	Need to manage backend environment and configuration	No backend to manage

Size:

Serverless is more fine-grained than microservices. A single microservice equals several serverless functions.

Communication:

Microservices in Microservice Architecture are evoked and communicate with each other via a lightweight communication protocol (usually HTTP API). Serverless Architecture is event-driven where functions are evoked as responses to certain events [11].

Serverless vs Containers:

Microservice Architecture usually uses containers instead of virtual machines for finer-grained execution environments and better isolation in-between components. This allows for component cohabitation and faster initialization and execution speed. However, it is the developers' responsibility to maintain and manage the infrastructure. Serverless Architecture, on the other hand, shifts this responsibility to third-party cloud providers and allows developers to focus more on development [12]. These will be compared in more detail in section 4.2.

Scalability:

Microservice Architecture allows the independent scaling of microservices to cope with variable demands on the system [13]. Serverless Architecture, on the other hand, allows the independent scaling of nanoservices, which are a more fine-grained unit of scale compared to microservice [11].

Testing:

The higher the granularity, the more complicated integration testing becomes and the harder it is to debug, troubleshoot and test [14]. Microservice Architecture has lower granularity than Serverless Architecture and thus, allows for easier debugging, troubleshooting, and testing. Additionally, as the microservices approach is mature, it is well supported by tools, processes, and necessary documentation [14].

Time to Market:

The time Serverless Architecture takes to deploy new features to production is greatly reduced because of its lightweight programming model and operational ease. This also means that prototypes can be quickly created and demonstrated to investors or clients [14]. For Microservice Architecture, when applying new features, the system has to be carefully planned and designed so that the existing application does not break. As opposed to just creating new functions in serverless applications for new features, microservices applications require decision-making as to how to add said new features (i.e. should we modify an existing microservice or should we create microservice). This decision-making process is time-consuming [15].

Cost:

Microservice requires personnel capable of configuring, maintaining, and managing the backend infrastructure which may incur additional costs. A Serverless Architecture shifts the backend management responsibility from application developers to third-party cloud vendors. This approach reduces overall project costs via bypassing hiring, and onboarding expenses of specialised personnel as well as storage costs and hardware investments [16].

4.2 Serverless vs Containers

Containers are a solution to the problem of how to get the software to run reliably when moved from one computing environment to another. A container 'contains' both an application and all the elements the application needs to run properly, including system libraries, system settings, and other dependencies [17]. Table II provides a brief comparison between Serverless and Container architecture.

Table II: Difference between Container and Serverless

Characteristics	Container	Serverless
Cost	Even though no one is using it, the company still needs to pay for it	Pay-as-you-go
Scalability	- Autoscale in range of number of containers specified prior to deployment - Need to determine the number of containers to be deployed in advance	Autoscale
Start-up and Maintenance	Developers take care of server management and initial set up time is high	The vendor takes care of server management
Provider	- Does not depend on third-party provider plans - No limit on operation duration - Can use any programming languages we want	- Risk of potential vendor lock-in - Limited operation duration - Can only use programming languages supported / compatible with the vendor services
Consistency and Testing	Run the same no matter where they are deployed, so easy to test in a local environment	Hard to replicate on a local environment, so hard to test locally

Cost:

Containers are constantly running even when no one is using it. This incurs unnecessary costs to the companies. Serverless functions, on the other hand, are only created and executed when a request is made and companies would only be charged based on the function run time. Therefore, they only need to pay for what they use [14].

Scalability:

Serverless Architecture can handle unexpected load spikes easily. On the other hand, containers can not and developers need to determine the number of containers to be deployed in advance [18]. If the demand is higher than the pre-specified number of containers, it would not be possible to produce more containers [14].

Start-up and Maintenance:

Containers take longer to set up initially than serverless. Developers have to configure system settings, and libraries to set up containers. Developers also need to manage, update, and maintain each container they deploy. While for serverless, the third-party vendor takes care of all management and software updates for the servers that run the code[18].

Provider:

A container doesn't depend on third-party plans and doesn't risk a potential vendor lock-in. Due to its nature, containers are not limited and bottlenecked by vendors' requirements to runtime, storage space, and RAM.
Serverless functions, by design, are short-term processes that should not take up a lot of RAM. Therefore, it has a limited number of requests and operation duration [14]. For example, AWS Lambda maximum execution time is 15 minutes [19]. In addition, serverless functions support few programming languages and are highly dependent on the specific vendor [14]. Serverless functions are also often cloud provider-specific, which can be problematic when switching vendors. Thus, companies need to thoroughly investigate the capabilities and offerings of the chosen vendor as the vendors might not offer what the company wants.

Consistency and Testing:

Containers run the same no matter where they are deployed [18]. Therefore it is relatively simple to test a container-based application before deploying it to production. Serverless environments are hard to replicate on a local environment and so it is more difficult to test serverless web applications.

5. HISTORICAL CONTEXT

5.1 System Architecture

The overall historical trend of system architecture revolves around optimizing the software in aspects such as efficiency, maintainability, usability, portability, reliability, security, etc.
Initially, applications were deployed on dedicated / bare-metal servers. These servers are single-tenant physical servers that are not shared between multiple users. Thus, running just 1 application on 1 server resulted in extremely inefficient resource utilization.
To improve on this, machine virtualization was introduced. Multiple applications are grouped and isolated on a virtual machine instead of occupying the entire server. The unit of scale thus changed from a whole physical server into a virtual machine which occupies part of the physical server.

Still, virtual machine images are quite heavy as each virtual machine contains a full copy of an operating system. Thus, we would want to limit as much as possible the amount of redundant operating system code that needs to be stored. Hence, instead of using a virtual machine for an application, containerization was developed so that multiple applications can be deployed and run using the same operating system in a virtual machine. Each container would still contain the code and the dependencies of an application; however, the redundant operating system code is no longer repeated compared to having 1 application per virtual machine.

Nonetheless, there remains an issue in microservices architecture and that is at least one service instance always needs to be on for each microservice. This would not be as useful and resource-efficient if we are not using or seldomly use that microservice. Thus, the need for an architecture where service instances would only be executed when there is an actual request brought forth the concept of Serverless Architecture. Serverless Architecture (also known as FaaS) allows applications to run in "stateless compute containers that are event-triggered, ephemeral (may only last for one invocation), and fully managed by a third party". Faas allows us to execute code in response to events via "functions", which are only processed when there are active requests. Moreover, these so-called functions are even more fine-grained compared to the normal microservice, thus giving it the name nanoservice, and therefore, are even better for resource utilization and ease of deployment.

Overall, we moved from a single whole deployment unit to multiple small independent deployment units. Then, we went from having an application per server to having an application per virtual machine to then having multiple applications per virtual machine (containerization) [20].

5.2 Software Architecture

Traditionally, software applications would be built as a single unit of execution and deployment. Modules within are not independently executable nor independently deployable. Additionally, the lack of module boundaries in such systems results in degrading modularity overtime which makes modifying the code base extremely hard in later stages of development.

Thus, came Service-Oriented Architecture (SOA). In this architecture, we are starting to see clearer boundaries in-between the modules of an application. Within a SOA application, a service black-boxes its implementation through an interface and provides functionalities to other services via passing messages in-between. From here, it is observable that the amount of dependency in-between modules has been greatly reduced compared to traditional monolithic architecture. Nonetheless, these applications, though composed of distributed services, are not "distributedly" deployable. Thus, SOA applications are "conceptually monolithic from the deployment point of view".

Rectifying the mistakes of SOA, the Microservice Architecture was created. Each microservice is independently deployable, operable, and scalable. Specifically, there have been 3 generations of Microservice Architecture [21]:

- **1st generation** – Container orchestration: Individual services were packed into containers, and then deployed and managed via a container orchestration tool. Service discovery – the ability to detect and keep track of the location of other services within the system – and failure-handling mechanisms were implemented directly within the service's code. In this generation, service discovery and fault-handling are the responsibility of the service. This makes locating services hard when the number of services within the application keeps increasing. Additionally, as new services can also be implemented using a different programming language, reusing existing fault-handling code is not often possible.

- **2nd generation** – Service discovery and fault tolerance: To improve on the drawbacks from the 1st generation, this generation introduced discovery services and fault-tolerant communication libraries. Services within the system would all register themselves under the same common discovery service. An analogy is that instead of a person trying to remember and locate another person from their knowledge/memory, they make use of an "all-knowing GPS" service that everyone registers under and knows everyone's location (the phonebook would be another great example). Thus, it is observable that the responsibility of locating other services has shifted from the service itself to the discovery service. Fault-tolerant communication libraries, on the other hand, made possible code reusability as all services would delegate fault-handling to a suitable communication library. This has again taken off more responsibility from the services themselves. However, this is not without drawbacks. As these libraries become more complicated, catering them for multiple different languages becomes hard. Thus, developers tend to have to implement new services in only languages supported by the libraries. This limits the freedom and potential of the services themselves and goes against the autonomy principle of microservices, where one can freely choose any programming language to use if they are the most appropriate.

- **3rd generation** – Sidecar and service mesh: This generation tries to further improve reusability as well as addressing the autonomy issue back in generation 2. Service proxies, or sidecars, were introduced and served as the intermediate of communication between a service and its caller. Before, even though we were using third-party libraries to manage our communication, traffic, and service discovery, we still had to write some code within each of our services to invoke those libraries. However, via encapsulating all of those similar functionalities into a sidecar, we can now re-use that sidecar for all the other services (and the sidecar technically becomes a traffic management, fault-tolerant communication, and service discovery service in itself). This greatly reduces the amount of redundant code as we do not need to repeat the configuration code in each of the services. Moreover, sidecars can be used by modules written in other languages as long as those modules know how to make a request to the sidecar (i.e. we do not need to code a sidecar for each language we have). This addresses the autonomy issue from the 3rd generation.

Thus, we see an underlying trend where software developers strive to improve ease of maintainability, deployment (through increasing modularity), and reusability of applications. We went from an extremely coupled massive single application to a multi-service application but still monolithic in terms of deployment to a multi-service application where each service is also independently deployable.

The emergence of the Serverless Architecture also rides the same trend. The need to manage the infrastructure of the application is delegated to external platforms which reduce further responsibility from the application and us developers.

A newer type of serverless computing, FaaS, further polishes the fine-grained modularity discipline from microservices via the concept of nanoservices. A nanoservice is a short-lived function that can be created, replaced, deleted quickly and easily, and run only when evoked. An example of a nanoservice would be a function to validate if a student is a student of a project where the microservice is the Student class. Thus, instead of having an always-on microservices, FaaS provides nanoservices that would only be created (i.e. take up resources) when they are required [20], [22].

6. COST AND BENEFITS OF SERVERLESS ARCHITECTURE

6.1 Costs

To gain a better understanding of the costs associated with FaaS, a cost comparison has been made with IaaS. To compare these two service types, the services provided by the leading cloud provider, AWS, will be used as a comparison point. In particular, a comparison will be made between AWS's FaaS service, AWS Lambda, and AWS's IaaS service, EC2. The bills associated with using both of these services have been calculated for the given situation:

> A university tool provided for students enrolled in FIT3170 is currently being tested. Within this unit, there are 5 projects with each project having 5 groups working on it. Each group currently has 5 members within them.
>
> The tool currently has three functions and, on average, each member is expected to send a request to each of these functions 20 times throughout the day. Over a 30-day month, this equates to 75,000 requests in total.

For AWS Lambda, we will assume that each request has a runtime of 1000ms or 1s when allocated a memory capacity of 3008MB. Using AWS's Lambda pricing calculator, this service would incur a total cost of $3.68USD ($5.06AUD) per month [60].

For AWS EC2, we will assume that for our application, it uses the smallest possible system requirements of 1 vCPU with a memory capacity of 1 GiB and uses a 1 GB of storage memory using the General Purpose SSD. Using AWS's pricing calculator, the service would incur a total cost of $6.23 USD ($8.57AUD) per month [61].

This means that AWS Lambda would be ~40% cheaper per month than AWS EC2 even though we are selecting the smallest possible system requirement for the EC2 instance but overestimating the amount of processing required to perform a Lambda function. However, if the number of requests was doubled, the costs associated with Lambda would become $7.37 USD ($10.14AUD) which is higher than that of EC2. FaaS would, therefore, be more efficient than IaaS until a certain amount of requests are made.

6.2 Benefits and Disadvantages

Operational:

Serverless Architecture, as stated previously, removes the need to directly manage infrastructure when building and deploying applications. This provides potential operational benefits as it eliminates the need to manage physical components such as the hardware being used as well software-related components such as the virtual machine operating system or web server software processes [23]. Capacity provisioning is also handled through automatic scaling based on the call frequency to the endpoint. This reduces the amount of configuration overhead required when deploying and running code. This abstraction from operational concerns improves the development speed of software as the effort and time previously spent on managing the deployment infrastructure can instead be placed into enhancing the system.

Nonetheless, infrastructure management does pose a concern as vendor lock-in can occur. Vendor lock-in occurs when an application becomes completely dependent on a single third-party service provider. With vendor lock-in, factors such as your system's availability and platform operations costs are completely dependent on the vendor's discretion [24]. This dependency raises issues as it can be both costly and time consuming to move your application to a new platform or a new service provider.

Resource Efficiency:

FaaS is typically provided using a pay-as-you-go pricing model. This payment model means a cost is only incurred when a request occurs based on its execution time. As opposed to the usual payment model based on active time, this optimizes usage costs as applications are typically not being requested continuously [25]. In combination with the automatic capacity provisioning, over-provisioning resources are, therefore, no longer required with Serverless Architecture. Furthermore, this allows for zero capacity scaling where no cost is incurred if no requests occur, reducing unnecessary costs [20].

This pricing model, however, can become costly for certain applications. As a cost is incurred only when a request is made and for the length of its duration, this price model is targeted for functions that receive small amounts of requests and have small execution times. Pay-as-you-go pricing model would, therefore, be inefficient for high traffic applications that are expected to receive many requests or have long run times [24]. Double spending is also another concern when using this pricing model. When a function calls upon another function during its execution, the additional function would also fall under this pricing model. This can result in unexpected costs if functions have high coupling.

Development and Deployment:

Compared to Monolithic Architecture, Serverless Architecture is more flexible in how applications are composed. Rather than developing a complete large application, FaaS can be composed of smaller individual, autonomous segments known as functions. These functions can then be deployed on a FaaS provider as separate smaller components rather than one monolithic application. This subdivided structure helps during development as it is easier to debug and update these segmented components compared to a large system [25]. Furthermore, this would be more resource-efficient as only the function needs to be called when that functionality is required compared to having the entire system being active for a singular request.

However, Serverless Architecture introduces multiple new complexities to the system. Integration can become more difficult as more components now have to be considered. As the architecture also uses stateless computing, it is not possible to preserve state across functions calls [20]. Additionally, Serverless Architecture introduces potentially new bugs to the system as unexpected issues can appear when integrating components or when deploying the code onto the service platform itself.

Security:

As Serverless Architecture removes the need to maintain the backend infrastructure, escalated privileges such as admin rights or SSH access permissions are no longer necessary. This reduces multiple avenues of security risks as these administrative rights are only known by the providers. The physical security and internal patching of software are also handled by these third party providers which remove unnecessary operations security overhead. Furthermore, granular IAM rights can be applied to these individual segments which reduce unintentional access permissions as functions can only access what they need [26].

However, Serverless Architecture also introduces new security risks which will be explained in section 8.2 below.

7. SERVERLESS ARCHITECTURE AND DEVOPS

With the growth of Serverless Architecture, DevOps has certainly been impacted by it. The management and maintenance of an application's infrastructure can be considered a part of DevOps. As Serverless Architecture eliminates this requirement, this aspect of DevOps has certainly been reduced. However, that is not true for the entirety of DevOps.

DevOps refers to a combination of practices and philosophies that bridges the development and operations teams that previously worked in siloes. This results in the merging of the two teams where the merged team now functions across the whole software development cycle[59]. DevOps allows teams to be more agile so that they can build, test and deploy application features in increments making the process of software development faster and more reliable[27].

One of the benefits of using serverless computing is that it simplifies the responsibilities of the Ops component of DevOps[28].

This is because teams do not need to worry about provisioning and maintaining the servers themselves while using Serverless Architecture as this administrative overhead will be taken care of by the cloud service providers. Though, it is to be noted that it is still the responsibility of DevOps to set up and configure pipelines that will enable the developers to build, test and deploy the application to these serverless platforms [29]. The work associated with DevOps would decrease only after the serverless services have been set up by them.

Another challenge that these serverless applications introduce is that they are quite often highly coupled with third party cloud services; this means that debugging the application locally would be complicated as the serverless services often do not provide access to the execution infrastructure [28].

In essence, our finding was that the Serverless Architecture does not completely eliminate DevOps, but it reduces the amount of effort required to deploy, provision, and maintain servers. This provides an opportunity for enterprises to shift their prime focus to the business logic that DevOps was introduced for [30].

8. RISKS OF SERVERLESS ARCHITECTURE

Although Serverless Architecture assists developers in creating applications by shifting server-side responsibilities away from the developers, the architecture does not completely prevent risks to the organization. This is despite the server-side security provided by services such as AWS, MS Azure and GCP. There have been numerous examples of cloud security breaches whilst using services provided by these organizations.

AWS has had many instances of 'leaky' S3 buckets which are databases that store information for cloud-based applications. These databases can contain sensitive information such as hotel guest data, medical data, etc. This was a problem which made AWS implement security measures just to prevent accidental S3 data leaks [31]-[35].

These kinds of breaches are not just limited to AWS, with both MS Azure and Google Cloud also having examples of these breaches [36]-[40].

The cost of these data breaches can be high, with an average of $3.92 million in 2019 estimated by IBM. This is in contrast with cloud misconfigurations, which includes serverless architecture misconfigurations, that alone on average cost $4.41 million which is nearly $500,000 above the average cost of data breaches [41], [42]. Due to the large costs that data breaches can incur even with the security provided by serverless services, it is important to address the risks to Serverless Architectures. Additionally ,there are several drawbacks to utilizing Serverless Architectures which can be divided between non-security and security-related risks.

8.1 Non Security Risks

When developing with Serverless Architectures, there can be many different types of non-security risks. A few of these non-security risks include Vendor control, Multi-tenancy problems, Vendor lock-in, Repetition of logic across client platforms, Loss of server optimizations and No in-server state for Serverless FaaS [5].
Vendor control is the outsourcing of the application to a third-party service which gives the vendor who owns the service some level of control over the system. These include "system downtime, unexpected limits, cost changes, loss of functionality, forced API upgrades, etc." Since developers are relying on these services to host their applications an unexpected system downtime for the vendor will also affect the applications hosted. Additionally, sudden change in cost, limits and functionality may cause the developer to migrate to other solutions and have the application be turned offline [43].

The multitenancy problems are the issues with having multiple users using the same machine on different instances which is a balance between security, robustness and performance. Multitenancy issues can include dealing with users from different countries under different laws which exposes these users to attackers using the same machine. Although this problem is not unique to Serverless Architectures, some serverless services provide solutions for multitenancy with AWS being one of the most common. These problems will need to be considered before developers design multitenancy solutions despite being deployed with a serverless service [44], [45].

Vendor lock-in is where the developer's application becomes too reliant on the Serverless Architecture so that the APIs or the services the code uses will be unable to operate without running on the specific vendor. This would make it difficult for example to transfer from AWS to MS Azure if the API relies on AWS specific functions [46].

Repetition of logic across client platforms is where code for different applications is replicated and therefore all requires maintenance when one of the applications needs to be changed. This can happen for example when BaaS is implemented which causes all developer code to exist in the frontend which means duplicate methods when communicating with the backend for several applications [47].

Utilizing BaaS makes the developers lose server optimization capabilities which may impact client performance. This is because in a serverless service the vendor controls the backend supplied to the developer and not the developers themselves. A certain method to assist with this issue is to incorporate FaaS [48].
Since FaaS typically has no control over when the host containers for the functions start and stop. This means that there is no in-server state for Serverless FaaS, which makes an external file or database store or out-of-process cache a requirement for serverless platforms when managing the state. These will be slower and in terms of out-of-process caches may cause inefficiencies [49].

8.2 Security Risks

Besides the non-security-related risks, there are many inherent security risks with developing using Serverless Architectures. These can be split between the new attack vectors and the critical risks. The new attack vectors are new vulnerabilities that are introduced by Serverless Architectures which include: Increased attack surface, Attack surface complexity, and Overall system complexity.

Due to the increase in the amount of data required for serverless functions, an increase in the overall attack surface can be observed. This is because the number of event sources such as APIs, storage and other devices communicating with the serverless application also increases.

The Serverless Architecture is relatively new which can make it difficult for the developers to understand the attack surface of their application and misconfigured it to be insecure. Since the applications developed with Serverless Architectures are a relatively new type of software environment, it can, therefore, be difficult to visualize and monitor [50]. It is important for developers to utilize enough logs and troubleshooting of events and functions when an attack on the serverless system occurs. This is because it can provide valuable information on how to prevent future intrusions and potentially recover from a successful breach.

It is commonly agreed that the SAS top 10 are the biggest risks to Serverless Architecture Security [51]. These are listed in Table III:

Table III: Critical Risks related to Serverless Architectures [52], [53]

Serverless Architecture Security (SAS) Code	Vulnerability Name
SAS-1	Function Event Data Injection
SAS-2	Broken Authentication
SAS-3	Insecure Serverless Deployment Configuration
SAS-4	Over-Privileged Function Permissions and Roles
SAS-5	Inadequate Function Monitoring and Logging
SAS-6	Insecure 3rd Party Dependencies
SAS-7	Insecure Application Secrets Storage
SAS-8	Denial of Service and Financial Resource Exhaustion
SAS-9	Serverless Function Execution Flow Manipulation
SAS-10	Improper Exception Handling and Verbose Error Messages

SAS-1 — Function Event Data Injection

Similar to the OWASP top 10, Injection attacks are the most common risks in applications and Serverless Architecture is no exception. This is caused due to Unsanitized Input being passed into the server on the cloud, which allows for relatively common attacks such as SQL Injection or any other form of data injection that can be processed by the server. The serverless environment cannot reliably protect the business logic of your application unless
sanitization of input from the developer is applied [54], [56].

SAS-2 — Broken Authentication

Broken Authentication is also in the OWASP top 10 as the number 2 threat to applications both with and without servers. However, Serverless Architectures promote Microservice Architectures which usually require more public APIs which themselves need robust authentication schemes. Therefore, it is recommended that developers utilize the serverless environment's authentication facilities rather than developing their own [55], [56].

SAS-3 — Insecure Serverless Deployment Configuration

Serverless environments allow for customization of settings which can induce security flaws in the environment itself. A common mistake is the incorrect configuration of cloud storage authentication/authorization [56].

SAS-4 — Over-Privileged Function Permissions and Roles

Serverless applications are composed of functions that require privileges to operate in the serverless environment. When a function is given too many permissions and roles it can be exploited to be used to perform actions that should not be authorized to users that have access to that function. This is the reason why applications, including serverless applications, should follow the principle of “least privilege” to minimize the potential of functions overreaching their intended capabilities [56].

SAS-5 — Inadequate Function Monitoring and Logging

The importance of function monitoring and logging is to react appropriately to a potential breach in application security. Since serverless applications are located under a vendor rather than the developers, host-based security controls cannot be implemented because it is under the control of the vendor. Despite the vendor providing logging, it may not always be sufficient for the organization and so function monitoring and logging may need to be implemented [56].

SAS-6 — Insecure 3rd Party Dependencies

Although serverless functions may be secure, the 3rd party dependencies that they can utilize may be insecure. Imported vulnerable 3rd party packages should only be used in small pieces of code doing discrete tasks to minimize the risk [56].

SAS-7 — Insecure Application Secrets Storage

A secrets storage is where sensitive data such as API keys, Database credentials and Encryption keys reside. Insecure Application Secrets Storage is where the secret storage is not secure, such as being stored in plain text as environment variables. Encrypting these secret storages and making sure they are decrypted only during function execution is important to secure the application secrets storage [56].

SAS-8 — Denial of Service and Financial Resource Exhaustion

Denial of Service (DoS) attack is a common method over recent years to prevent access to the application by overloading the network. These methods of attacks prevent the application from properly responding to legitimate users. Various methods exist when performing a DoS on serverless applications, an example includes a specialised boundary string which is an extremely inefficient regular expression that when executed by AWS Lambda can cause the application to freeze until the function times out. To prevent this, implementing efficient serverless functions and setting environment variables that limit the potential impact of a DoS attack should be done [56].

SAS-9 — Serverless Function Execution Flow Manipulation

Serverless Function Execution Flow Manipulation is the manipulation of application flow that can subvert application logic and potentially bypass security measures that are in place. This is due to the application relying on functions that are executed in a specific order which may not always be enforced properly. The best way to deal with this risk is to not make assumptions about legitimate invocation flows and make sure that the proper permissions are set for the functions [56].

SAS-10 — Improper Exception Handling and Verbose Error Messages

Although exception handling and error messages are good aspects of developing applications, improper exception handling and verbose error messaging are the problem where exceptions are handled by having error messages with excessive information which can be used by attackers to gain information on the application. This information can
then be used to assist in attacks on the application especially if it reveals it is an application with known exploits. To address this issue, it is recommended to use the Serverless Architecture services to avoid verbose debug printing [56].

Overall, employing a Serverless Architecture does not prevent risk from occurring and requires extensive assessment to weigh the threat and impact. Although there are many benefits from utilizing Serverless Architectures, especially when it comes to shifting backend responsibilities away from the developers, there are several inherent risks associated with Serverless Architectures. However, by following appropriate methods to deal with the above risks, it can be beneficial for many developers to choose a Serverless Architecture.

9. PROJECT SUITABILITY

Due to the nature of the Serverless Architecture, not all projects are suited for development with a serverless service. In general, the Serverless Architecture is best used for applications that have short execution times, and low traffic or short instances of high traffic based on the calculations from section 6.1.

A major advantage that Serverless Architectures have is their convenience to developers to deploy applications into production. This makes a serverless solution appealing to projects that are likely to undergo continuous development and need to deploy new features quickly.

When the load for the application can not be determined beforehand, being able to allocate and adjust the computational resources on the run is a huge benefit that the Serverless Architecture supports. This helps to save money because the customer only pays for the runtime of the services that they used [57].

Serverless Architecture is suitable for IoT devices which usually produce multiple short bursts of data from sensors between long idle times. So the Serverless Architecture's pay-as-you-go pricing model suits it better compared to having a dedicated server which inflicts costs even when it is idle [58].

Since serverless applications are meant to handle tasks with short execution times, it should be used for applications that do not require a large amount of processing. For example, AWS Lambda has a limit on the maximum execution time of 15 minutes, with a pre-set maximum execution time of 5 minutes. Therefore, complex calculations are not meant to be run on serverless systems [19].

FaaS should not be considered for data-security sensitive applications. While these third-party service providers typically provide heavy emphasis on the security of data, these platforms are not immune to data breaches [32]. Currently, when using serverless approaches, the security team of an application has to handle the complexity of securing data on the third-party service. This additional layer of complexity leads to cases where data breaches have occurred due to developer inexperience with the serverless services and occasionally the third-parties themselves causing these leaks [35], [39].Therefore, security-critical systems that contain sensitive data would not be suitable for Serverless Architecture.

10. RECOMMENDATIONS

10.1 Recommendation for FIT3170

We would recommend using Serverless Architectures in a future FIT3170 Project because of the following reasons:

- Serverless Architecture shifts the need to maintain and monitor our infrastructure to third-party FaaS platforms. This can be tedious and complicated, especially for students who lack prior experience in provisioning infrastructure on cloud platforms. Thus, students would not need to worry about backend environment configuration and settings, allowing them to focus on developing features.

- Based on our FIT3170 experience, there are certain components in our project where Serverless Architecture would have made the development process less expensive and easier for us. Our Agile team worked on the SPMD project. There were periods of time where we did not modify or use the deployed codebase for days but we still incurred costs during those idle periods. In contrast, FaaS would provide the pay-as-you-go pricing model which would optimize cost expenditure since a cost only incurs when a request occurs instead of paying for the services even when they are idle.

- FIT3170 projects are essentially prototypes, a proof of concept to verify an idea. Based on our research, serverless applications remove the overhead required when deploying code, making it easier.

However, there are some reasons for concern when it comes to adopting a Serverless Architecture:

- Since FaaS is not mature enough, it does not have as many tools for performing long term maintenance such as automated testing tools. On the other hand, microservices have more tools and processes that can support a project for a longer time.

- Vendor lock-in may occur on the project if the client is too reliant on the cloud provider which may make it hard to reuse client-side code on different vendors. If the project is intended to be used outside of the original vendor then it may require drastic changes to the code. This can be an issue when handing over certain FIT3170 projects to clients who might want to use the application with a different vendor.

Based on our experience, we would recommend using the Serverless Architecture in FIT3170 for developing applications. However, careful consideration still needs to be taken by individual teams on whether Serverless Architecture is suitable for their project.

10.2 Recommendation for other units

Similarly, we recommend that Serverless Architecture should be considered for FIT3077. As this unit teaches students the different existing software engineering architecture and designs principles, a discussion of Serverless Architecture coincides with this unit's learning outcomes. FIT3077 provided students with an understanding of the different types of design principles and software architectures. This helps students in making informed decisions for developing robust and maintainable software applications. However, Serverless Architectures is not currently taught by FIT3077.

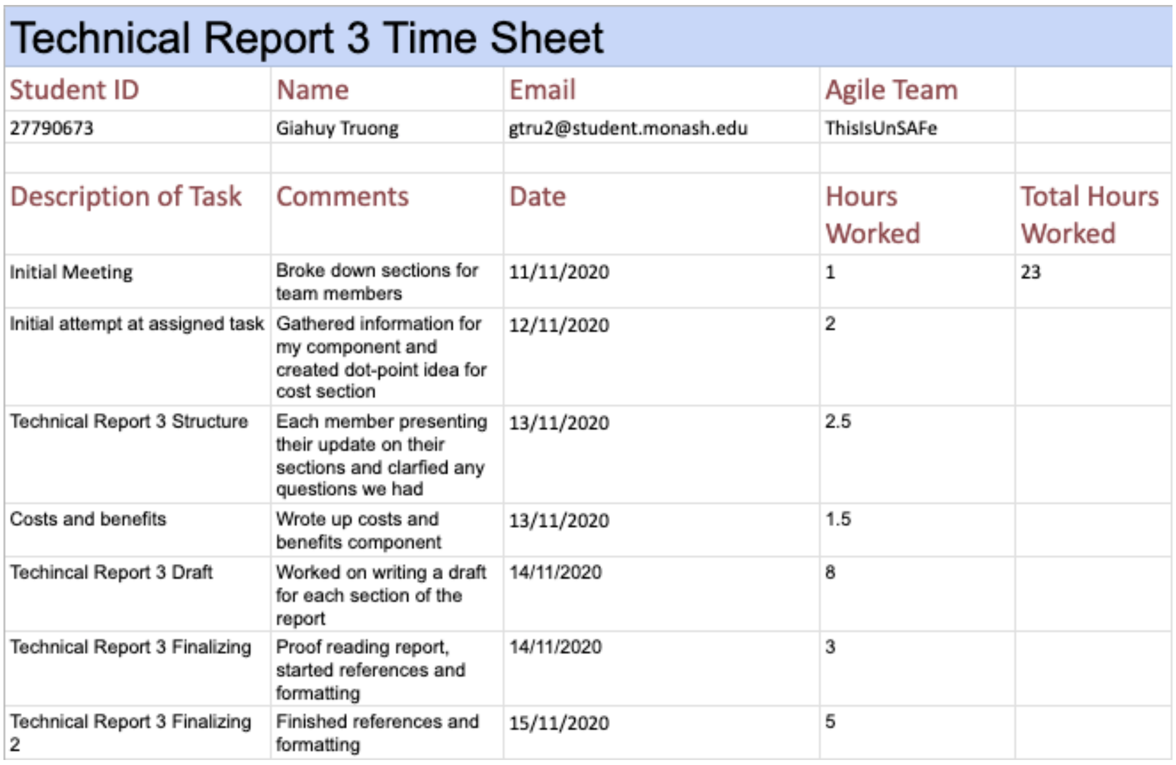
To stay updated with the current I.T. industry, software developers and engineers need to be knowledgeable about the prevalence of Serverless Architectures in order to develop software applications according to industry standards. However, having only discussions on Serverless Architectures in FIT3077 may not be entirely useful, since there would not be much time spent on practical experience related to Serverless Architectures. Nonetheless, it should still be worthwhile to discuss this in brief so that students are aware of this software architecture so they can make more informed judgements.

Another unit that our team came across while searching units that are suitable to discuss this concept was Distributed Computing (FIT3142). According to the Monash handbook, this unit covers concepts such as distributed cloud computing, clusters, web services, and grids [62]. Based on these learning outcomes, we recommend a discussion on Serverless Architecture will be a useful addition to this unit. However, as this is not a core unit for the Software Engineering students, none of our team members previously had enrolled into this unit. Hence, the recommendation for this unit is made solely on the basis of the unit overview and learning outcomes listed on the Monash handbook.

Overall, based on our judgement and experience, we recommend that these two additional units FIT3077 and FIT3142 could incorporate a discussion on Serverless Architecture into their curriculum.

11. APPENDICES

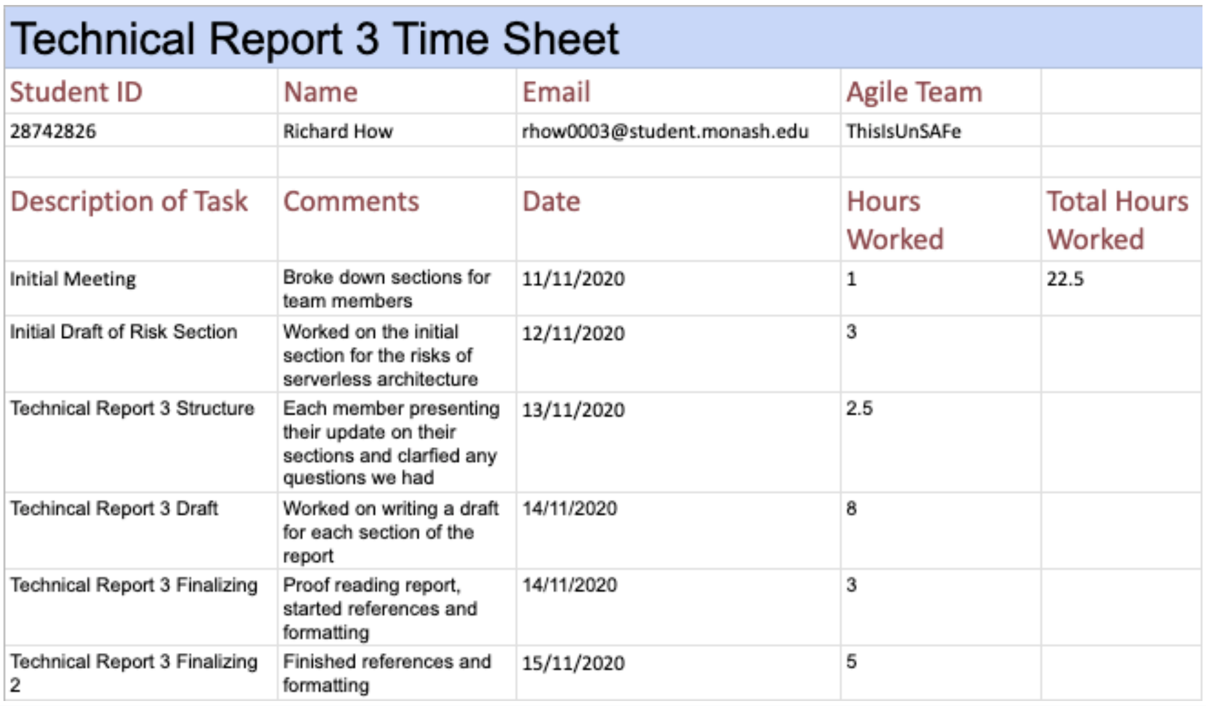
Appendix 1: Giahuy Truong Timesheet



Technical Report 3 Time Sheet

Student ID	Name	Email	Agile Team	
27790673	Giahuy Truong	gtru2@student.monash.edu	ThisIsUnSAFe	
Description of Task	Comments	Date	Hours Worked	Total Hours Worked
Initial Meeting	Broke down sections for team members	11/11/2020	1	23
Initial attempt at assigned task	Gathered information for my component and created dot-point idea for cost section	12/11/2020	2	
Technical Report 3 Structure	Each member presenting their update on their sections and clarfied any questions we had	13/11/2020	2.5	
Costs and benefits	Wrote up costs and benefits component	13/11/2020	1.5	
Techincal Report 3 Draft	Worked on writing a draft for each section of the report	14/11/2020	8	
Technical Report 3 Finalizing	Proof reading report, started references and formatting	14/11/2020	3	
Technical Report 3 Finalizing 2	Finished references and formatting	15/11/2020	5	

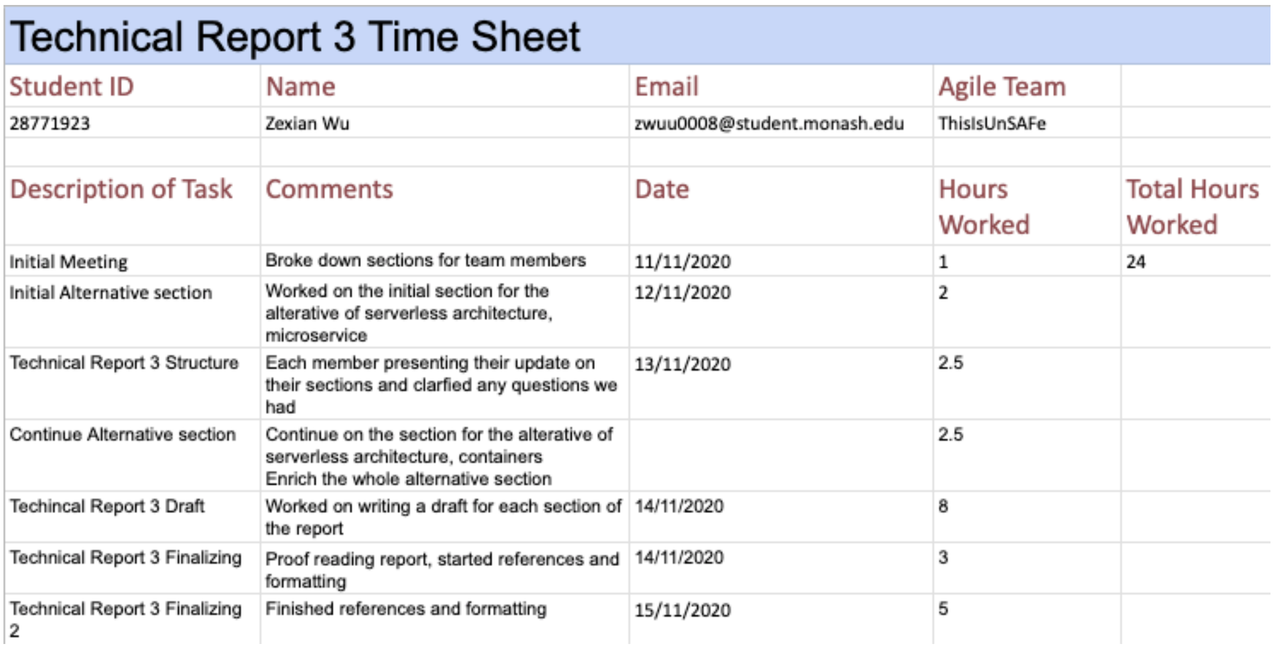
Appendix 2: Richard How Timesheet



Technical Report 3 Time Sheet

Student ID	Name	Email	Agile Team	
28742826	Richard How	rhow0003@student.monash.edu	ThisIsUnSAFe	
Description of Task	Comments	Date	Hours Worked	Total Hours Worked
Initial Meeting	Broke down sections for team members	11/11/2020	1	22.5
Initial Draft of Risk Section	Worked on the initial section for the risks of serverless architecture	12/11/2020	3	
Technical Report 3 Structure	Each member presenting their update on their sections and clarfied any questions we had	13/11/2020	2.5	
Techincal Report 3 Draft	Worked on writing a draft for each section of the report	14/11/2020	8	
Technical Report 3 Finalizing	Proof reading report, started references and formatting	14/11/2020	3	
Technical Report 3 Finalizing 2	Finished references and formatting	15/11/2020	5	

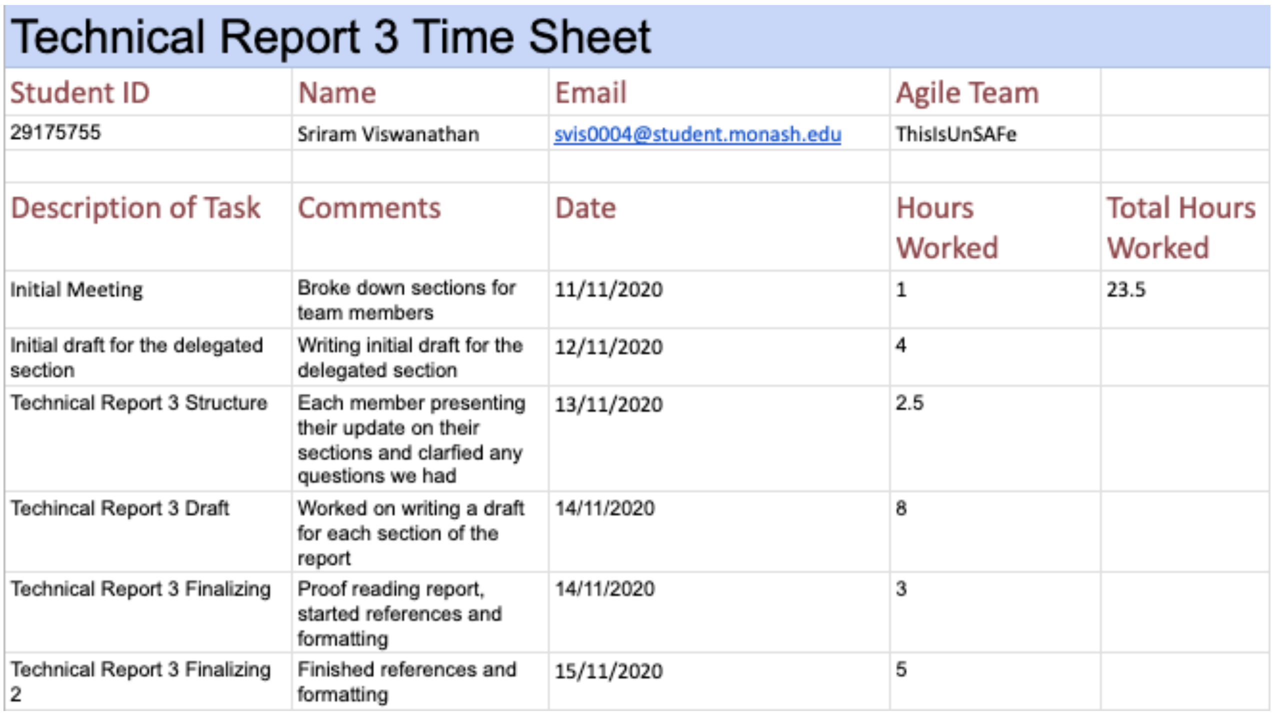
Appendix 3: Zexian Wu Timesheet



Technical Report 3 Time Sheet

Student ID	Name	Email	Agile Team	
28771923	Zexian Wu	zwuu0008@student.monash.edu	ThisIsUnSAFe	
Description of Task	**Comments**	**Date**	**Hours Worked**	**Total Hours Worked**
Initial Meeting	Broke down sections for team members	11/11/2020	1	24
Initial Alternative section	Worked on the initial section for the alterative of serverless architecture, microservice	12/11/2020	2	
Technical Report 3 Structure	Each member presenting their update on their sections and clarfied any questions we had	13/11/2020	2.5	
Continue Alternative section	Continue on the section for the alterative of serverless architecture, containers Enrich the whole alternative section		2.5	
Techincal Report 3 Draft	Worked on writing a draft for each section of the report	14/11/2020	8	
Technical Report 3 Finalizing	Proof reading report, started references and formatting	14/11/2020	3	
Technical Report 3 Finalizing 2	Finished references and formatting	15/11/2020	5	

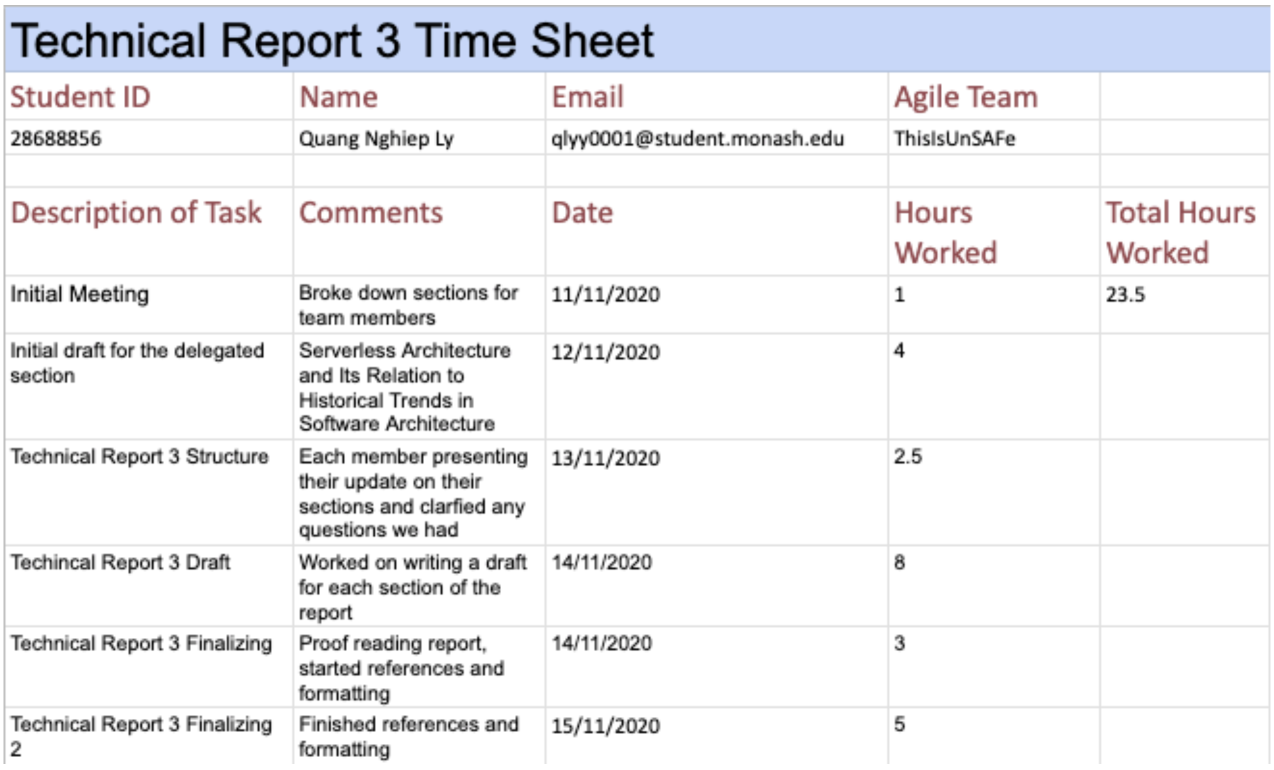
Appendix 4: Sriram Viswanathan Timesheet



Technical Report 3 Time Sheet

Student ID	Name	Email	Agile Team	
29175755	Sriram Viswanathan	svis0004@student.monash.edu	ThisIsUnSAFe	
Description of Task	**Comments**	**Date**	**Hours Worked**	**Total Hours Worked**
Initial Meeting	Broke down sections for team members	11/11/2020	1	23.5
Initial draft for the delegated section	Writing initial draft for the delegated section	12/11/2020	4	
Technical Report 3 Structure	Each member presenting their update on their sections and clarfied any questions we had	13/11/2020	2.5	
Techincal Report 3 Draft	Worked on writing a draft for each section of the report	14/11/2020	8	
Technical Report 3 Finalizing	Proof reading report, started references and formatting	14/11/2020	3	
Technical Report 3 Finalizing 2	Finished references and formatting	15/11/2020	5	

Appendix 5: Quang Nghiep Ly Timesheet



Technical Report 3 Time Sheet

Student ID	Name	Email	Agile Team	
28688856	Quang Nghiep Ly	qlyy0001@student.monash.edu	ThisIsUnSAFe	
Description of Task	Comments	Date	Hours Worked	Total Hours Worked
Initial Meeting	Broke down sections for team members	11/11/2020	1	23.5
Initial draft for the delegated section	Serverless Architecture and Its Relation to Historical Trends in Software Architecture	12/11/2020	4	
Technical Report 3 Structure	Each member presenting their update on their sections and clarfied any questions we had	13/11/2020	2.5	
Techincal Report 3 Draft	Worked on writing a draft for each section of the report	14/11/2020	8	
Technical Report 3 Finalizing	Proof reading report, started references and formatting	14/11/2020	3	
Technical Report 3 Finalizing 2	Finished references and formatting	15/11/2020	5	

12. BIBLIOGRAPHY

[1]"The Benefits of Serverless Computing and its Impact on DevOps | Hacker Noon", Hackernoon.com, 2020. [Online]. Available: https://hackernoon.com/the-benefits-of-serverless-computing-and-its-impact-on-devops-e75d82c47ac4. [Accessed: 14- Nov- 2020].

[2]"Get hands-on experience with Serverless Computing using IBM Cloud Functions", IBM Developer, 2020. [Online]. Available: https://developer.ibm.com/series/get-hands-on-with-serverless-series-page/. [Accessed: 12- Nov- 2020].

[3]"Serverless Computing – Amazon Web Services", Amazon Web Services, Inc., 2020. [Online]. Available: https://aws.amazon.com/serverless/. [Accessed: 14- Nov- 2020].

[4]"Serverless Architecture Market Size 2020-2026 | Industry Share Report", Global Market Insights, Inc., 2020. [Online]. Available: https://www.gminsights.com/industry-analysis/serverless-architecture-market. [Accessed: 11- Nov- 2020].

[5]"Serverless Architectures", martinfowler.com, 2020. [Online]. Available: https://martinfowler.com/articles/serverless.html. [Accessed: 14- Nov- 2020].

[6]"What is Function-as-a-Service (FaaS)?", Red Hat, 2020. [Online]. Available: https://www.redhat.com/en/topics/cloud-native-apps/what-is-faas. [Accessed: 13 - Nov- 2020].

[7]"How Serverless will Change DevOps | Scalyr", Scalyr, 2020. [Online]. Available: https://www.scalyr.com/blog/how-serverless-will-change-devops. [Accessed: 9- Nov- 2020].

[8]"What are microservices?", Red Hat, 2020. [Online]. Available: https://www.redhat.com/en/topics/microservices/what-are-microservices. [Accessed: 9- Nov- 2020].

[9]"Microservices", Red Hat, 2020. [Online]. Available: https://www.redhat.com/en/topics/microservices. [Accessed: 9- Nov- 2020].

[10]"When should I choose between serverless and microservices?", SearchAppArchitecture, 2020. [Online]. Available: https://searchapparchitecture.techtarget.com/answer/When-should-I-choose-between-serverless-and-microservices. [Accessed: 13- Nov- 2020].

[11]"Serverless and Microservices: a match made in heaven?", Medium, 2020. [Online]. Available: https://medium.com/@PaulDJohnston/serverless-and-microservices-a-match-made-in-heaven-9964f329a3bc. [Accessed: 11- Nov- 2020].

[12]"Serverless vs. Microservices architecture: what does the future of business computing look | ByteAnt", Byteant.com, 2020. [Online]. Available: https://www.byteant.com/blog/serverless-vs-microservices-architecture-what-does-the-future-of-business-computing-look/. [Accessed: 14- Nov- 2020].

[13]"Monolith Vs Microservice Vs Serverless — The Real Winner? The Developer | Hacker Noon", Hackernoon.com, 2020. [Online]. Available: https://hackernoon.com/monolith-vs-

microservice-vs-serverless-the-real-winner-the-developer-8aae6042fb48. [Accessed: 12- Nov- 2020].

[14]"Serverless vs. Microservices: What you need to know for cloud - Ahead in the Clouds", Computerweekly.com, 2020. [Online]. Available: https://www.computerweekly.com/blog/Ahead-in-the-Clouds/Serverless-vs-Microservices-What-you-need-to-know-for-cloud. [Accessed: 11- Nov- 2020].

[15]"Microservices vs. Serverless", Fathomtech.io, 2020. [Online]. Available: https://fathomtech.io/blog/microservices-vs-serverless/. [Accessed: 14- Nov- 2020].

[16]"Serverless Vs Microservices Architecture - A Deep Dive | Hacker Noon", Hackernoon.com, 2020. [Online]. Available: https://hackernoon.com/serverless-vs-microservices-architecture-a-deep-dive-lw2u3w0b. [Accessed: 13- Nov- 2020].

[17]P. Rubens, "What are containers and why do you need them?", CIO, 2020. [Online]. Available: https://www.cio.com/article/2924995/what-are-containers-and-why-do-you-need-them.html. [Accessed: 9- Nov- 2020].

[18]"Serverless computing vs. containers | How to choose", Cloudflare, 2020. [Online]. Available: https://www.cloudflare.com/learning/serverless/serverless-vs-containers/. [Accessed: 12- Nov- 2020].

[19]"AWS Lambda enables functions that can run up to 15 minutes", Amazon Web Services, Inc., 2020. [Online]. Available: https://aws.amazon.com/about-aws/whats-new/2018/10/aws-lambda-supports-functions-that-can-run-up-to-15-minutes/. [Accessed: 11- Nov- 2020].

[20]N. Kratzke, "A Brief History of Cloud Application Architectures", Applied Sciences, vol. 8, no. 8, p. 1368, 2018.

[21]P. Jamshidi, C. Pahl, N. Mendonca, J. Lewis and S. Tilkov, "Microservices: The Journey So Far and Challenges Ahead", IEEE Software, vol. 35, no. 3, pp. 24-35, 2018.

[22]"Evolution up to Serverless Architecture - DZone Cloud", dzone.com, 2020. [Online]. Available: https://dzone.com/articles/evolution-up-to-serverless-architecture-1. [Accessed: 14- Nov- 2020]

[23]"What is Serverless Architecture?", Twilio.com, 2020. [Online]. Available: https://www.twilio.com/docs/glossary/what-is-serverless-architecture. [Accessed: 9- Nov- 2020].

[24]A. Yigal, "Should You Go 'Serverless'? The Pros and Cons - DevOps.com", DevOps.com, 2020. [Online]. Available: https://devops.com/go-serverless-pros-cons/. [Accessed: 9- Nov- 2020].

[25]"Comparing Serverless Architecture Providers: AWS, Azure, Google, IBM, and Other FaaS Vendors - DZone Cloud", dzone.com, 2020. [Online]. Available: https://dzone.com/articles/comparing-serverless-architecture-providers-aws-az. [Accessed: 10- Nov- 2020].

[26]"What is Serverless Security | Check Point Software", Check Point Software, 2020. [Online]. Available: https://www.checkpoint.com/cyber-hub/cloud-security/what-is-serverless-security/#:~:text=The%20fact%20that%20serverless%20functions,a%20fantastic%20opportunity%20for%20security. [Accessed: 14- Nov- 2020].

[27]"What is DevOps | Atlassian", Atlassian, 2020. [Online]. Available: https://www.atlassian.com/devops/what-is-devops. [Accessed: 14- Nov- 2020].

[28]"Why You Should Go Serverless for DevOps", Stackify, 2020. [Online]. Available: https://stackify.com/why-you-should-go-serverless-for-devops/. [Accessed: 14- Nov- 2020].

[29]B. Gain, V. Haggar, M. Vizard and J. Mathews, "Why Serverless Needs DevOps - DevOps.com", DevOps.com, 2020. [Online]. Available: https://devops.com/why-serverless-needs-devops/#:~:text=With%20serverless%20architecture%2C%20there%20is,operations%20functions.%E2%80%9D%20Challa%20said. [Accessed: 14- Nov- 2020].

[30]Moduscreate.com, 2020. [Online]. Available: https://moduscreate.com/blog/will-serverless-kill-the-devops-star/?fbclid=IwAR16UBuEtAqzx6Mpjw87oQ-JNkFMAjgmCSoL6PeTTYkXBOI3hd27l3eOQTM#:~:text=So%2C%20no%2C%20serverless%20won',Repository%20for%20new%20innovative%20solutions. [Accessed: 13- Nov- 2020].

[31]"Leaky AWS S3 buckets are so common, they're being found by the thousands now – with lots of buried secrets", Theregister.com, 2020. [Online]. Available: https://www.theregister.com/2020/08/03/leaky_s3_buckets/. [Accessed: 12- Nov- 2020].

[32]E. Chickowski, "Leaky Buckets: 10 Worst Amazon S3 Breaches", Businessinsights.bitdefender.com, 2020. [Online]. Available: https://businessinsights.bitdefender.com/worst-amazon-breaches. [Accessed: 11- Nov- 2020].

[33]"Data on millions of hotel guests exposed in cloud storage leak | WeLiveSecurity", WeLiveSecurity, 2020. [Online]. Available: https://www.welivesecurity.com/2020/11/10/data-millions-hotel-guests-exposed-leak/. [Accessed: 12- Nov- 2020].

[34]"Leaky AWS S3 Bucket Once Again at Centre of Data Breach | News", Gurucul, 2020. [Online]. Available: https://gurucul.com/news/leaky-aws-s3-bucket-once-again-at-centre-of-data-breach. [Accessed: 13- Nov- 2020].

[35]C. Cimpanu, "AWS rolls out new security feature to prevent accidental S3 data leaks | ZDNet", ZDNet, 2020. [Online]. Available: https://www.zdnet.com/article/aws-rolls-out-new-security-feature-to-prevent-accidental-s3-data-leaks/. [Accessed: 9- Nov- 2020].

[36]E. Kedrosky and P. Sornson, "Microsoft Leaks 250M Customer Details in Azure Fat-Finger Faux Pas - Security Boulevard", Security Boulevard, 2020. [Online]. Available: https://securityboulevard.com/2020/01/microsoft-leaks-250m-customer-details-in-azure-fat-finger-faux-pas/. [Accessed: 14- Nov- 2020].

[37]"Microsoft Azure Cloud Data Leak: User Error? - MSSP Alert", MSSP Alert, 2020. [Online]. Available: https://www.msspalert.com/cybersecurity-breaches-and-attacks/microsoft-azure-data-leak/. [Accessed: 10- Nov- 2020].

[38]C. Cimpanu, "Microsoft discloses security breach of customer support database | ZDNet", ZDNet, 2020. [Online]. Available: https://www.zdnet.com/article/microsoft-discloses-security-breach-of-customer-support-database/. [Accessed: 10- Nov- 2020].

[39]K. Yedakula, "Unsecured Microsoft Azure Blob Exposes Millions of Automatic Number Plate Recognition Images | Cyware Hacker News", cyware-social-nuxt, 2020. [Online]. Available: https://cyware.com/news/unsecured-microsoft-azure-blob-exposes-millions-of-automatic-number-plate-recognition-images-9b04c528. [Accessed: 12- Nov- 2020].

[40]"Pharma Giant Pfizer Leaks Customer Prescription Info, Call Transcripts", Threatpost.com, 2020. [Online]. Available: https://threatpost.com/pharma-pfizer-leaks-prescription-call-transcripts/160354/. [Accessed: 13- Nov- 2020].

[41]C. Bradford, "7 Most Infamous Cloud Security Breaches - StorageCraft", StorageCraft Technology Corporation, 2020. [Online]. Available: https://blog.storagecraft.com/7-infamous-cloud-security-breaches/. [Accessed: 12- Nov- 2020].

[42]"How much does a data breach cost you? - CloudSEK", CloudSEK, 2020. [Online]. Available: https://cloudsek.com/how-much-does-a-data-breach-cost-you/. [Accessed: 14- Nov- 2020].

[43]"Serverless Computing—a New Business' Little Helper", Medium, 2020. [Online]. Available: https://medium.com/sciforce/serverless-computing-a-new-business-little-helper-e85f87734f59. [Accessed: 10- Nov- 2020].

[44]"The Drawbacks of Serverless Architecture - DZone Cloud", dzone.com, 2020. [Online]. Available: https://dzone.com/articles/the-drawbacks-of-serverless-architecture. [Accessed: 11- Nov- 2020].

[45]"AWS Lambda vs Azure Functions vs Google Cloud Functions: Comparing Serverless Providers", Simform.com, 2020. [Online]. Available: https://www.simform.com/aws-lambda-vs-azure-functions-vs-google-functions/. [Accessed: 13- Nov- 2020].

[46]"Serverless vendor lock-in: Should you be worried? | TechBeacon", TechBeacon, 2020. [Online]. Available: https://techbeacon.com/enterprise-it/serverless-vendor-lock-should-you-be-worried. [Accessed: 12- Nov- 2020].

[47]"Serverless Architecture", Medium, 2020. [Online]. Available: https://codecraft.medium.com/serverless-architecture-ade9dbaac1f5. [Accessed: 12- Nov- 2020].

[48]"Serverless Architecture", Medium, 2020. [Online]. Available: https://medium.com/@nipunaprashan/serverless-architecture-3d5996e54e2f. [Accessed: 9- Nov- 2020].

[49]Ijrter.com, 2020. [Online]. Available: https://www.ijrter.com/papers/volume-4/issue-4/going-serverless-a-review.pdf. [Accessed: 11- Nov- 2020].

[50]"How Serverless Architecture is Changing Security | Distillery", Distillery, 2020. [Online]. Available: https://distillery.com/blog/serverless-architecture-security/. [Accessed: 14- Nov- 2020].

[51]"Top 10 Security Risks in Serverless Architectures - DZone Security", dzone.com, 2020. [Online]. Available: https://dzone.com/articles/top-10-security-risks-in-serverless. [Accessed: 11- Nov- 2020].

[52]C. Osborne, "The top 10 security challenges of serverless architectures | ZDNet", ZDNet, 2020. [Online]. Available: https://www.zdnet.com/article/the-top-10-risks-for-apps-on-serverless-architectures. [Accessed: 10- Nov- 2020].

[53]"The 12 Most Critical Risks for Serverless Applications", Cloud Security Alliance, 2020. [Online]. Available: https://cloudsecurityalliance.org/blog/2019/02/11/critical-risks-serverless-applications/. [Accessed: 9- Nov- 2020].

[54]"Event Injection: Protecting your Serverless Applications - Jeremy Daly", Jeremy Daly, 2020. [Online]. Available: https://www.jeremydaly.com/event-injection-protecting-your-serverless-applications/. [Accessed: 13- Nov- 2020].

[55]"OWASP Top Ten Web Application Security Risks | OWASP", Owasp.org, 2020. [Online]. Available: https://owasp.org/www-project-top-ten/. [Accessed: 14- Nov- 2020].

[56]"puresec/sas-top-10", GitHub, 2020. [Online]. Available: https://github.com/puresec/sas-top-10. [Accessed: 14- Nov- 2020].

[57]"Serverless Architecture: When To Use It and What Benefits You Get", Apiko | Learn, 2020. [Online]. Available: https://apiko.com/blog/serverless-architecture-benefits/. [Accessed: 9- Nov- 2020].

[58]"Using IoT with Serverless to Tackle Global Issues", Medium, 2020. [Online]. Available: https://medium.com/thundra/using-iot-with-serverless-to-tackle-global-issues-9695a40d5d8d#:~:text=There%20is%20no%20more%20need,cloud%20services%20to%20cloud%20vendors. [Accessed: 10- Nov- 2020].

[59]"What is DevOps? - Amazon Web Services (AWS)", Amazon Web Services, Inc., 2020. [Online]. Available: https://aws.amazon.com/devops/what-is-devops/. [Accessed: 15- Nov- 2020]

[60]"AWS Lambda – Pricing", Amazon Web Services, Inc., 2020. [Online]. Available: https://aws.amazon.com/lambda/pricing/. [Accessed: 14- Nov- 2020].

[61]"AWS Pricing Calculator", Calculator.aws, 2020. [Online]. Available: https://calculator.aws/#/estimate. [Accessed: 15- Nov- 2020]