

# A comprehensive Secure Serverless Container-based Architecture (SSCAR)

MSc Research Project  
Cloud Computing

**Kamrun Nahar Ali**  
Student ID: 21139474

School of Computing  
National College of Ireland

Supervisor: Mr. Vikas Sahni

**National College of Ireland**  
**MSc Project Submission Sheet**  
**School of Computing**



**Student Name:** Kamrun Nahar Ali  
**Student ID:** 21139474  
**Programme:** MSc. Cloud Computing **Year:** 2022-2023  
**Module:** MSc Research Project  
**Supervisor:** Mr. Vikas Sahni  
**Submission Due Date:** 15/12/2022  
**Project Title:** A comprehensive Secure Serverless Container-based Architecture (SSCAR)  
**Word Count:** ...6445..... **Page Count** ...22.....

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Kamrun Nahar Ali.....

**Date:** 12/12/2022.....

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
<b>Attach a Moodle submission receipt of the online project submission,</b> to each project (including multiple copies).	<input type="checkbox"/>
<b>You must ensure that you retain a HARD COPY of the project,</b> both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# A comprehensive Secure Serverless Container-based Architecture (SSCAR)

Kamrun Nahar Ali  
21139474

## Abstract

Due to the inherent flexibility and scalability, serverless computing in combination with microservice deployment is emerging as the most promising and ever-growing service available today. Public cloud service provider's serverless computing, also known as Function-as-service enables developers to build an application without having to worry about infrastructure. However, certain obstacles exist on public cloud platforms, such as vendor lock-in, computing constraints, regulatory constraints, and security vulnerabilities. As a result, there is increased interest in deploying serverless computing on private infrastructure. Containers are one of the most popular techniques to create serverless computing enabling the use of an existing framework. Microservice architecture is worth the exposure due to its ability to expand quickly with minimized cost and high reliability is essential. With the ease of development, containers can bring some serious security threats to application owners. This research has implemented a serverless event-driven container-based (with docker container) framework in Azure with added security steps such as RBAC, image scanning, and identity verification which allows a safe container image run. The evaluation's findings demonstrated that the SSCAR architecture can make it simple to deploy customized serverless event-driven functionalities while managing and monitoring the Azure generated logs. This research has identified a niche for secure container-based serverless Azure framework applications that can be deployed to any company segment interested in moving to the cloud and experimenting with a cost-effective solution without having to be concerned about vendor lock-in.

## 1 Introduction

Data holds the key to the future and that's why data processing has become crucial for having useful data. Cloud computing has made processing tasks simpler by offering on-demand storage, networking, and computing resources. Serverless cloud computing was introduced by AWS Lambda back in 2014 and it was later adopted by Microsoft in 2016. Serverless adds an abstract layer in the cloud paradigm where server-side works are discharged from the developers (Hassan et al., 2021).

Provisioning and using services have not been as easier rather has a certain background to it. Let us look back and try to understand the concept of virtualization before moving ahead. Virtualization is a pivotal aspect of cloud computing and its services, which provides numerous

virtual machines, networks, and storage available with a click. With Hypervisor technology<sup>1</sup>, the software allows multiple operating systems to launch and run using a single hardware system (Bermejo and Juiz, 2022). The traditional virtual machine (VM) tend to be heavier in cost and time as VM technology uses software to simulate a hardware system to allocate and isolate computing resources and thus manages resources for cloud computing for different users. The hypervisor or virtual machine technology is having a major drawback in independence and resource contention because the same hardware's performance degrades each time the VM runs a copy of the OS. This is an overhead of creating and maintaining a VM in software development.

PaaS clouds must handle the necessity of packaging and application management. A solution must be based on technologies that enable secure, portable, and interoperable sharing of the underlying platform and infrastructure in a virtualized environment. The lightweight virtualization technology which enables rapid distribution and deployment of the application is called Containerization (Simonsson et al., 2021). Applications are stored in containers as packaged, autonomous, and ready-to-deploy components, as well as, if necessary, middleware and business logic in the form of binaries and libraries. Over the Operating System virtualization, containerization adds an abstraction layer where all standardized software, programs, and dependent libraries are bundled and encased (Pahl, 2015). The new age of serverless computing is getting its limelight in today's cloud world due to its advantages of zero administrative work, elasticity, and minimum costs. On the other hand, containerization is a virtualization technology that addresses the issue of resource sharing and program independence. The run-time environment of the application is the main feature of containers. As compared to traditional virtualization, this is more flexible.

The need for security increases as cloud development methodologies expand with novel concepts because the data and system are exposed on the network and subject to attack at any time. With added security steps this virtualization method provides a lightweight solution and horizontal scalability. Containers give developers more control over the environment the application runs in, and the languages and libraries used. Although these advantages come with additional maintenance. Because of this, containers are often very useful for migrating legacy systems to the cloud since replicating the application's original running environment is relatively easier.

Serverless architecture is not new to this area but the method of achieving a serverless event-driven secure framework varies from one to another. (Pérez et al., 2018) had discussed and proposed Serverless Container-aware Architecture (SCAR), a highly parallel, event-driven, and scalable architecture in AWS without any measures of handling a secure framework. This study has taken the serverless container-based architecture to implement in Azure and has ensured preventive security measures to enable and maintain the system's security. Minding the economic resource constraint, this study has aimed to provide a niche architecture to the business organization that can adapt this framework in their favour and transition to their cloud journey.

---

<sup>1</sup> <https://en.wikipedia.org/wiki/Hypervisor>

## 1.1 Research Question

To what extent a container-based event-driven scalable serverless framework can be secured and what measures are useful to build a secure reliable container-based secure serverless architecture?

## 1.2 Document Structure

Before delving into the intricacies of the proposed project, a quick review of research on the issue would aid and enlighten the subject matter, as different perspectives would aid in better understanding related themes and their problem areas. This paper is divided into six parts. Section 2 provides a Related Work section with literature from relevant research publications and gaps associated with it. Section 3 attempts to describe the methodology, technology stack, of this study and why the study adapted certain characteristics of virtualization and containerization as software and architecture specification respectively. Section 4 emphasizes the process of implementation of SSCAR in Azure. Section 5 reviewed the framework using four case studies and discussion. Section 6 closes this analysis by outlining areas for improvements.

## 2 Related Work

SaaS, IaaS, and PaaS are the three basic categories into which cloud computing services can be broadly categorized. The purpose of all cloud services is to shift the responsibility for the upkeep of the platform, infrastructure, or software. Addressing the issues of load-balancing, security, auto-scaling, and availability, helped pave the road for serverless computing (Jonas et al., 2019).

Amazon AWS, Microsoft Azure, and Google Cloud are the leading cloud computing providers<sup>2</sup>. Amazon AWS is the cloud's oldest operator, having seized the early bird advantage to dominate the cloud business industry. However, with its planned industry approach, Microsoft Azure intends to give AWS a run for its money. The creation of modern applications has been transformed by serverless architecture for some time. Because of the development of container-based technologies, their usability has risen with time. The industry has seen exponential growth in event-driven services with the use of AWS Lambda and Azure functions (Expósito Jiménez et al., 2018). AWS Lambda was the first of its kind in the case of serverless computing services and then Microsoft Azure Functions and GCP Cloud functions followed the trail.

A typical example of serverless computing is Function-as-a-service (FaaS) model, in which developers create a function in various languages, and the same is then deployed to cloud platforms and can be triggered based on events. Burkat et al. (2021) discussed and evaluated a relatively new service, Container-as-a-service. They highlighted the features of container-based virtualization. Serverless computing has been applied in various areas of computing including IoT (Cheng et al., 2019), ETL (Pogiatzis & Samakovitis, 2021), and video processing (Ao et al., 2018). Serverless technology has brought many benefits, but it has

---

<sup>2</sup><https://www.gartner.com/reviews/market/cloud-infrastructure-and-platform-services>

additional obstacles due to decreased quantitative measurement with limited observability and increased system complexity (Leitner et al., 2019; Lenarduzzi & Panichella, 2021). However, serverless computing is not all good it poses some serious challenges as well. On one side they provide security because of cloud providers on the other hand serverless cloud services create some unique threats and challenges as well which were discussed by Marin et al. ( 2022).

Azure Functions<sup>3</sup> provides a variety of interface methods including CLI, API, and GUI and uses related plugins for Visual Studio and Visual Studio Code to access the platform. Azure Functions' price structure is comparable to AWS Lambda's; however, it is based on the amount of memory that serverless functions use. Additionally, billing is executed with a minimum execution time of 100 milliseconds. Azure Functions has a 600-second maximum function execution timeout. The execution and administration unit in Azure Functions is the function app, which is still made up of several functions. There is no deployment package cap for Azure Functions, and its flexible memory allocation can accommodate up to 1,536 MB of RAM. The observability of Azure Functions is provided by Microsoft via Azure Application Insights. Azure functions uses three hosting tiers for serverless applications: consumer, premium, and dedicated. Microsoft offers a general-purpose Azure Marketplace for the market that consists of serverless applications (Wen et al., 2022).

As an alternative, method of in-built serverless services provided by Cloud Service Providers like AWS's Lambda or Azure Function or Azure Automation Beborrtta et al. ( 2020) discussed and demonstrated that python and its vast libraries have been widely used by Cloud providers to write serverless mechanism and frameworks; thus python is one of the most prominent languages to be adopted by various providers. The novelty behind the proposed architecture is to create a framework to use a serverless computing paradigm using python and its libraries for data processing. Another study discussed vendor lock-in issues in serverless computing as the providers give a very little scope of utilizing executable code systems and serverless computing becomes highly platform dependent because authentication, configuration management, storage, or monitoring becomes tightly coupled to the platform services (Adzic & Chatley, 2017) .

Developers have the option to wrap the execution environment, include the package and library dependencies, and source code inside a container with the help of containerization which is also termed as operating system virtualization. Unlike Virtual Machines (VMs), containers do not require an operating system simulation starting up. The starting up of a containerized application is faster than VMs (VasanthaKumari & Arulmurugan, 2022). To put into perspective the operating system, containers can be considered as software systems. Since the execution environments are bundled into a place, so the containers are considered very portable. Though containerization is a technology that has been around for a while, it has only begun to receive attention now. The possibility of becoming enslaved to a provider is a prominent concern highlighted in the conference paper of Becker Westphall & Olmsted ( 2016) and among IT executives regarding cloud computing. Mobility is constrained due to the high switching costs in terms of time and effort required to switch between cloud service providers.

---

<sup>3</sup> <https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview>

Containerization enables organizations to leverage several cloud providers to develop their desired applications (Kratzke, 2014). Abdelbaky et al. (2015) had shown in their experiment how an architecture made up of different cloud service providers can create a conglomerated robust architecture with the help of containerization technology. Another case-study on micro-service workloads highlighted the advantages of cloud and container clusters in scaling and deployment of application (Augustyn et al., 2022).

As cloud containers are gaining popularity, the question of how to keep them secure came into focus. Before docker containers had to run as a privileged user on the underlying OS, which made it possible for a root or administrator access to be obtained if certain components of the container were compromised, or vice versa (Watada et al., 2019). User namespaces, which allow containers to run as certain users, are now supported by Docker.

Deploying rootless containers is another choice to reduce access problems. These containers give an extra layer of security because they do not need root access. So, if a rootless container is compromised, the hacker will not have root access. The ability for several users to run containers on the same endpoint is another advantage of rootless containers. Docker currently supports rootless containers, but Kubernetes does not (Brasser et al., 2022).

The safety of images acquired from Docker Hub is another problem (Shu, Gu, and Enck, 2017; Kwon and Lee, 2020). Downloading a community-developed image does not necessarily ensure the security of a container. With the introduction of the Docker Content Trust feature in version 1.8, Docker began to solve this issue by confirming the identity of the image's publisher. Vulnerabilities can also be checked in images. This provides some assurance, but if containers are used for extremely sensitive applications, their verification processes might not be thorough enough. To guarantee that the security policies have been followed and updates have been made regularly, it would be prudent to produce the base image to maintain authenticity in this situation.

It can be concluded from the foregoing discussion, that there is a great deal of interest in serverless computing architectures for diverse scientific activities and web application development across the industry. But Serverless computing, however, there are very few architectures that propose a minimal effort with the maximum outcome. This study is going to demonstrate how a secure serverless container-based architecture can be modelled using cloud services to provide minimized processing time effective utilization of resources and thus reducing the overall cost of the framework.

## 2.1 Research Gaps

Table 1 summarizes the research gaps of five papers.

**Table 1**

Comparative Analysis of Related Work on Serverless Container based Architecture framework			
Research Reference	Approach	Benefits	Limitations
Serverless computing for container-based architectures (Pérez et al., 2018).	Created a serverless container architecture using AWS lambda for event-driven file processing.	AWS-based framework for event-driven serverless file processing.	No security measures were implemented and SCAR had not been implemented in other cloud services.
Serverless Computing: Economic and Architectural Impact (Adzic & Chatley, 2017).	Presented case studies showing how migrating to the cloud reduced hosting cost.	It motivates business to move to cloud.	The study is based on AWS services need more research on other cloud services.
The FaaS based Cloud Agnostic Architecture of Medical Services &mdash; Polish Case Study (Augustyn et al., 2022).	The case study provides an analysis of micro-service architecture for various workloads.	Highlights the advantages of cloud and container clusters in scaling and deployment of application.	The functional groups chosen for implementation in the FaaS model in this case study were utilized so infrequently.
Observability and chaos engineering on system calls for containerized applications in Docker (Simonsson et al., 2021)	Case study and observation by injecting system call failure in the containerized application and proposed framework for resiliency.	Proposes metrics and framework to improve observability of a container.	This approach provides observability of only applications' system call failure.
Serverless computing: a security perspective (Marin et al., 2022)	Discussed the various security flaws of serverless computing.	Provides an insight on possible security adversaries on Monoliths, Microservices and Serverless	The review paper shows adversaries can now launch Denial-of-Wallet or DoW attacks by flooding service requests, drastically increasing the cost for application owners.

## 3 Research Methodology

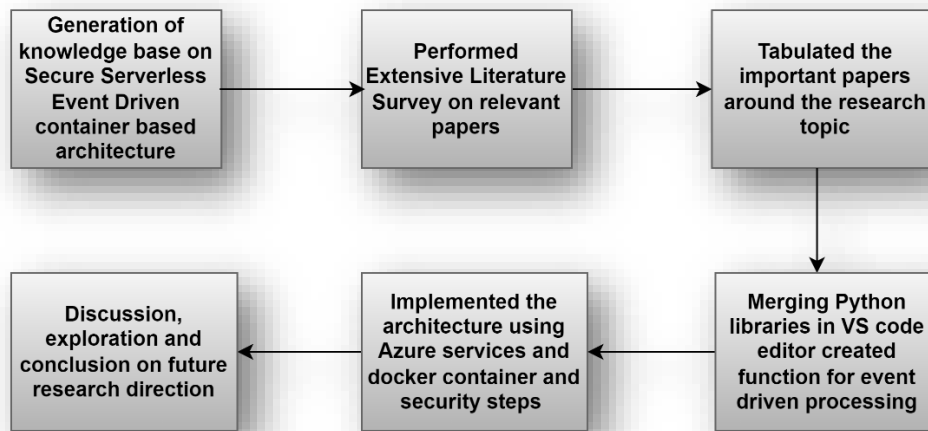
This section discusses the methodology of this study. The research process flow is given in 3.1 and 3.2, which provide an overview of the techniques and technologies used in this study.

### 3.1 Process Workflow

Figure 1 depicts the whole workflow of the study conducted on this topic. The first stage denotes the relevant study of the serverless computing paradigm from prior research work and other internet resources that contributed to the basic information for this project. The second stage suggests that the performed literature survey and critically evaluating various journals and articles around this area. The next stage is the tabulation of the performed survey to present



a better visualization and understanding of the works performed. Then the fourth and fifth step indicates the implementation process and finally at the sixth step this study concludes with a discussion and possible future work.



**Figure 1: Visual representation of research methodology behind SSCAR**

### 3.2 Technology stack

Table 2 lists various components of this study and in the following sections, the methodology of selecting those components has been discussed.

**Table 2**

Components	Tool Stack
<b>Cloud Service Provider</b>	Microsoft Azure
<b>Code Editor</b>	Visual Studio Code
<b>Operating System</b>	Windows
<b>Container Engine</b>	Docker
<b>Container Registry</b>	Docker Hub
<b>Programming Language</b>	Python
<b>Code Repository</b>	GitHub

**Cloud providers:** Selecting cloud service providers at a competitive price appears to be one of the hardest tasks in today’s industry, where businesses must take care of the existing system model and add a new segment to their business in order to grow faster and smother in the industry. Ayoub Kamal et al. (2020) provided an elaborate comparison in their study where they compared AWS, Azure, and GCP’s offerings and categorized their services into different tiers of pricing. They have highlighted storage capacity, location infrastructure, and computation which are some of the major service features that play a vital role when choosing cloud service providers for any business. From their study and finding its evident for this experiment in Azure was proven to be a better fit in terms of pricing and the goal of this study is to determine whether a secure serverless container-based architecture can be reliable enough to be used in the business segment. Also, Microsoft claims, Azure is one of the research-

friendly cloud service providers<sup>4</sup> in academia and encourages to use Azure in the cost of discovery.

**Code Editor:** According to a survey conducted by Mälardalen University Academy of Innovation Design and Engineering<sup>5</sup>, it was found that Visual Studio Code is highly popular among the developer community hence in this research the choice of code editor is Microsoft Visual Studio Code.

**Containers and Container Registry:** Containers provide more mobility, reduced start-up time, and higher resource usage than VMs, simplifying the creation and maintenance of large-scale cloud applications. Even though there are other alternative container engines, Docker is the most widely used container in cloud computing<sup>6</sup>. Clusters of containerized service instances serve as the foundation for microservice systems. The containers must be fault-tolerant, distributed, and highly available. (Khan, 2017) presented the capabilities of container tools and designed a framework that highlights the essential process for techniques to execute container orchestration. In an attempt to fill the gap of best orchestration tool, (Khan, 2017) and (al Jawarneh et al., 2019) found that Kubernetes performs well for complex orchestration deployment as compared to Apache Mesos<sup>7</sup>, Docker<sup>8</sup> but for its simpler solutions these orchestration tools are the best fit. Docker is a tool for packaging an application and all of its dependencies into a container. It accomplishes this by offering a set of tools and a consistent API for controlling kernel-level technologies including LXC containers, cgroups, and a copy-on-write filesystem. Docker uses AuFS (Advanced Multi-Layered Unification Filesystem) as its container filesystem. AuFS is a layered filesystem that can overlay one or more existing filesystems transparently (Scheepers, 2014). Docker can use AuFS to base containers on specific images. For example, an Ubuntu image may be used as the foundation for a variety of containers. This study is using Docker as a containerization tool as it aims to find a simple and cost-effective tool for secure serverless application deployment and development framework.

**Event-driven file processing using Python:** Python is the most popular coding language which has an extensive library set for several functionalities and is compatible with various platforms<sup>9</sup>. The programming model to achieve event-driven results in the following order:

1. Authorized user uploads file to azure blob.
2. The input file is made available to execute a container.
3. The script deletes the file from the input file.
4. The input file is processed and saved to an archived folder of the azure blob.
5. Generate Log for each file-processing event.

This approach is achieved by functions which detects the file and processes it using a python function that uses checkblob and moveblob user-defined library from the util package. The

---

<sup>4</sup> <https://edudownloads.azureedge.net/msdownloads/microsoft-azure-for-research-overview.pdf>

<sup>5</sup> <https://www.diva-portal.org/smash/get/diva2:1177860/FULLTEXT01.pdf>

<sup>6</sup> <https://www.g2.com/categories/container-engine>

<sup>7</sup> <https://mesos.apache.org/>

<sup>8</sup> <https://www.docker.com/>

<sup>9</sup> <https://pypi.github.io/PYPL.html>

python script processes the file and moves it to an output folder. Because this proposed framework manages data staging fundamentally, the developer only needed to shift their focus to how to process any file. Then, considering the limited storage space available, numerous instances of this script can be executed in parallel, each on its own invocation, to simultaneously process distinct files at scale. This method also makes testing easier because it can be done locally to process one file within a Docker container and then scaled out to thousands of concurrent invocations run on Azure.

## 4 Design Specification

The solution framework has been explained in the following subsection. 4.1 provides an overview of the model shares an insight into underlying methods and adapted best practices to build this model.

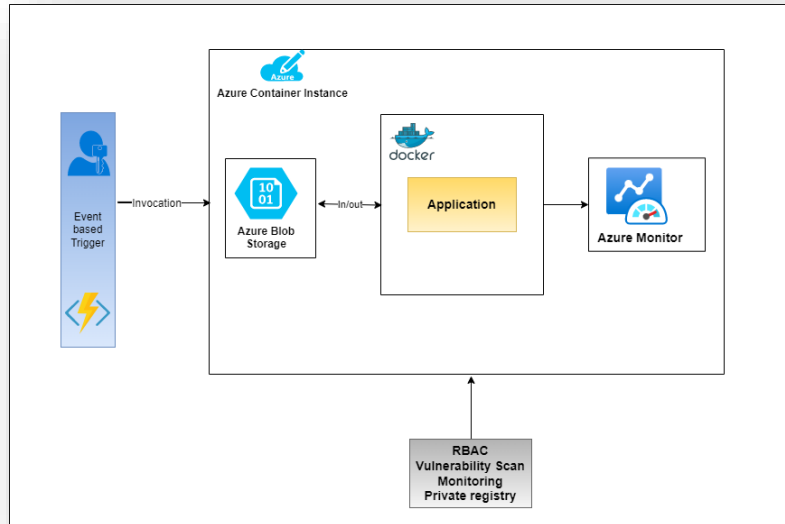
### 4.1 Architecture Overview

The proposed specification of the system design is demonstrated in Figure 2 which consists of four main components. Event-driven function, Serverless computing, Storage, and finally Security with monitoring log.

*Event-driven function:* Any trigger or event is a way to launch the function. The time-based, HTTP request, new entry in the database, or a new event in the system can act as a trigger to perform another task that can be used to detect events in the application. Bindings are used to connect a function to another resource for additional inputs or outputs. They support writing to a queue or a database, delivering an HTTP response, and a variety of other activities (Morenets & Shabinskiy, 2020). So, whenever there is a change detected by the containerized function in the blob storage, it identifies it as an event, detects the files and notifies to application for processing.

*Serverless Computing:* Serverless computing paradigm had been employed in this architecture using a python language and user-defined function where a file is read from input blob storage and the processed file is stored in a different location. Input then gets deleted by the system.

*Storage:* Azure blob storage stores the input and output files which can be accessed by only authorized users.



**Figure 2: Architecture of SSCAR**

*Security and Log Monitoring:* The entire system model gets encapsulated with its environment variables and dependencies into a docker container, and the instance is then pushed to the Azure Container instance. Azure Monitor keeps a log of the application activity. One security layer would be in Azure blob storage where Role-Based-Access-Control established, this will reduce data leakage and data breach. The second layer of security is to keep the docker container image private image and perform vulnerability scanning before pushing the image to the docker registry, which ensures network security. Docker file modification to create a non-root user while running the docker container image. This ensures that a container launch will establish a non-root user privileged image run and refrain any fatal modification to the image or root directory. Lastly, the system log with Azure monitor provides overall system health information.

*Containerization:* The storage requirements of the container instance are based on the size of the container image. The size of the container image has a direct impact on the amount of storage available to the container. Since the container image will be saved in the register, the size of the container image has an impact on the storage requirements of the container registry. Image size had been reduced using the following techniques:

- The image had been built from a scratch image.
- Used lightweight base image.
- Multistage built has been used.
- Reducing commands and concatenating into a single command for example in Figure 3 commands are directed in a single prompt.

```
RUN apt-get install -y example-package && rm -rf
variable/library/apt/list/*
```

**Figure 3**

In a small scale like this experiment, container size might be irrelevant but when dealing with a dynamic auto-scaling environment time to start the container can be translated as to waste of resources.

## 5 Implementation

Implementation of the framework has used below mentioned (Table 3) configuration of Azure and other services.

**Table 3**

Item	Resource
<b>Cloud Service Provider</b>	Microsoft Azure
<b>Subscription</b>	Azure for Students
<b>Region</b>	Norway-East
<b>Service Opted</b>	Azure Function, Monitor, Key Vault, Container Instance
<b>OS</b>	Windows
<b>Kernel</b>	Linux
<b>Image size</b>	1.5 GB memory, 0gpu, 1CPU
<b>Instance type</b>	Standard
<b>Network type</b>	Private
<b>Runtime Stack</b>	Python Version 3.9
<b>Consumption plan type</b>	Serverless
<b>Storage Redundancy</b>	Locally redundant storage
<b>Storage Performance</b>	Standard
<b>Container instance image</b>	QuickStart image
<b>Containerization tool</b>	Docker

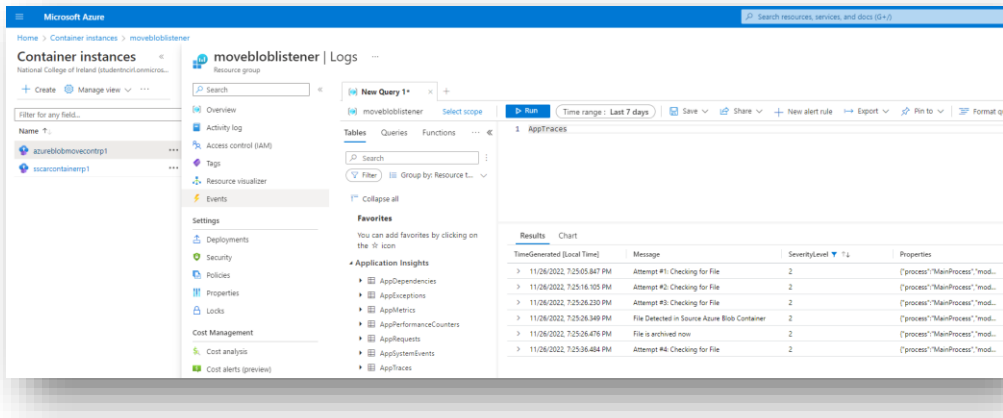
A python program has been written to check blob storage for files with checkblob and moveblob function libraries. This function acts as a listener and checks Azure blob storage for files and moves them upon finding them. The program performs event-driven file processing and generates user-readable logs when it detects the file and processes them to output blob storage. These logs were sent to Azure-application monitoring using AzureLogHandler package.

**Docker registry:** Web container configuration involves a methodical process. Writing a Docker file in a local environment is the first step. Then software packages that must be installed inside the Docker image in order for it to function are described in the text file, which





from the docker hub registry. One of the biggest bottlenecks in the performance of the container is the time taken to pull the image from the registry. Bigger the image; farther the location, greater the time taken to deploy the latest image from the registry. In this evaluation, it was clear that even after using a slim version of the OS image in the container, it took some time (~1.10 minutes) to kick in and executed the code. Figure 7 demonstrates a log of the Azure container instance.



**Figure 7: Container Instance Log**

- **Application runs on independent machine within the container**

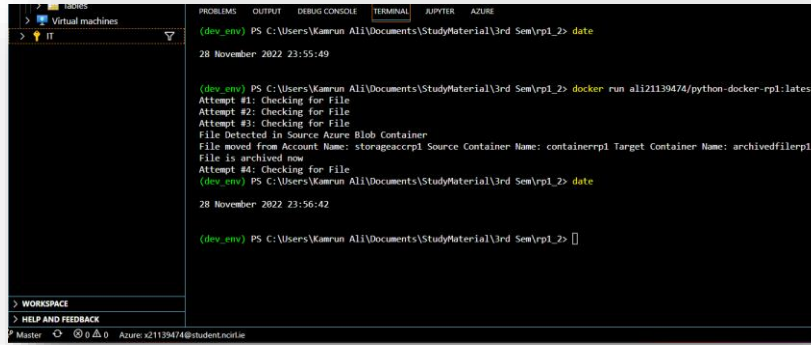
In order to replicate the 3-tier architecture or server, the local machine was considered as a server where the application is executed with basic configuration such as core i7 process, 16GB RAM, and 200GB free SSD storage. When executed locally, the container was executed within 53 seconds (Figure 8) which included manually placing the file to Azure Blob Storage. It must be noted that execution of container does not include time taken to build, pull the image from repository as it would be required only when there is a first-time setup or some changes in images.

## 6.2 Case Study 2: Operational Excellence

The capacity to run and monitor systems to create business value and enhance supporting processes and procedures is part of the operational excellence pillar. The amount of difficulty in implementing metrics that can ease the process of operations and monitoring has been assessed based on server-based vs cloud-based systems, as the method will be the same.

- Health Status
- Alert & logging mechanism





**Figure 8: Execution Status of docker container**

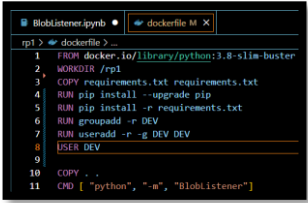
To generate health-related reports for the application, a server-based system will require an in-built utility in the program, or the app owner will need to write a custom script. A cloud-based system, on the other hand, has pre-defined health checks, logs, and other metrics to achieve operational excellence, such as reliability and availability. Furthermore, it enables users to create their own custom scripts to support any special use cases. Cloud-based health status checks also enable the creation of alerts that can run in real time and notify the user if a rule is violated (such as slow response of API, failure in connecting to app etc).

### 6.3 Case Study 3: Security

Based on pillars of container security outlined by (Brady et al., 2020), this case-study had been done to perform evaluate different Security aspects of modern-day threats. Table 4 suggests the considered evaluation parameters.

**Table 4: Security Parameter Evaluation**

Parameter	Local Server (Without Container)	Local Server (With Container)	Azure functions	Azure Container Instances
<b>Identity and Access Management</b>	Identity and access management in a 3-tier architecture is managed by on-premise infrastructure team. Even though users and application, now a days, can connect to Azure active directory and authenticate, restricted access rule needs to be set by Linux admin, within third party application, customize	One important aspect of container is to provide isolation to the application by running application inside an OS image. However, default behaviour of container is to run application using root user (full administrative access). This expands attack entire system with full privilege being exposed to intruders. This was easily managed by creating group and DEV user in docker file and run application using DEV user within container.	A priority-ordered allow or deny list regulates the network access to the app by configuring access limits. Subnets of the Azure Virtual Network or IP addresses may be in the list. There is an implicit deny all at the end of the list when there are one or more entries. One can easily define custom access	Azure Active directory, Various pre-defined access role can be used to define restricted user access. Also, similar improvements can be implemented as it was mentioned under Local Server (with container).

	<p>authorization in some cases which makes it tough to manage and has decent chances of missing out on security threat.</p>	 <pre> 1 FROM docker.io/library/python:3.8-slim-buster 2 WORKDIR /rp1 3 COPY requirements.txt requirements.txt 4 RUN pip install --upgrade pip 5 RUN pip install -r requirements.txt 6 RUN groupadd -r DEV 7 RUN useradd -r -g DEV DEV 8 USER DEV 9 10 COPY . . 11 CMD ["python", "-m", "BlabListener"] </pre> <p>This has ensured even if intruder get access to application, they will not be able to do further damage with this container user DEV. Other aspect is managing access to build infrastructure. It is important to ensure that CI/CD pipeline has limited access to build infrastructure so that intruder does not get access to application which may store password, token, secrets which can give access to sensitive information. This can easily be managed by CI/CD tool (Jenkins or Gitlab) which is widely used by many organizations.</p>	<p>restriction rules in Azure.</p>	
<p><b>Detective Controls</b></p>	<p><b>Code level scan:</b> Various third-party tools can be used to scan code vulnerabilities eg. SonarQube. However, it is app owner's responsibility to integrate such scan before migrating code to higher environments.</p>	<p><b>Code level scan:</b> Various third-party tools can be used to scan code vulnerabilities eg. SonarQube. However, it is app owner's responsibility to integrate such scan before migrating code to higher environments. <b>Container Scan:</b> Third party docker scanner can be used to scan vulnerabilities within the docker images. A static vulnerability scan on the docker image after building the image and before moving the image to the container instance. Dynamic scanning is also possible when image is pushed to the docker hub. However, this could be mostly under paid subscription.</p>	<p><b>Code level scan:</b> Various third-party tools can be used to scan code vulnerabilities e.g., SonarQube. However, it is app owner's responsibility to integrate such scan before migrating code to higher environments. <b>Container Scan:</b> Third party docker scanner can be used to scan vulnerabilities within the docker images. A static vulnerability scan on the docker image after building the image and before moving the image to the</p>	

			container instance. Dynamic scanning is also possible when image is pushed to the docker hub. However, this could be mostly under paid subscription.	
--	--	--	--	--

### 6.3.1 Limitations

Current scheme of testing and evaluation of security scope heavily relies on vulnerability scan count. While this option serves a good initial starting point for development and implementation process, an arbitrary number of vulnerabilities is not sufficient for production-level security. The default security policy utilized in this framework is at an initial phase and it would take additional paid security services which can meet all the security standards that a company must meet. Based on the results of this study it is evident that Docker images should pass at least a threat scanning and change very few changes in the file system.

### 6.4 Case Study 3: Reliability

Because of its core properties throughout the application's lifecycle, container architecture provides an advantage in terms of becoming reliable software. Change management, failure management, and workload management all contribute to reliability. In this study the cloud services chosen was able to handle availability based on availability zones and categories. This architecture was built with Locally Redundant Storage to keep resource constraints in mind, but it is suggested to use Geo Redundant Storage when developing application storage to boost storage dependability. This framework included Azure app insight, which was an excellent way to monitor and detect the health of the application and finally manage failure accordingly.

### 6.5 Case Study 4: Cost-Optimization

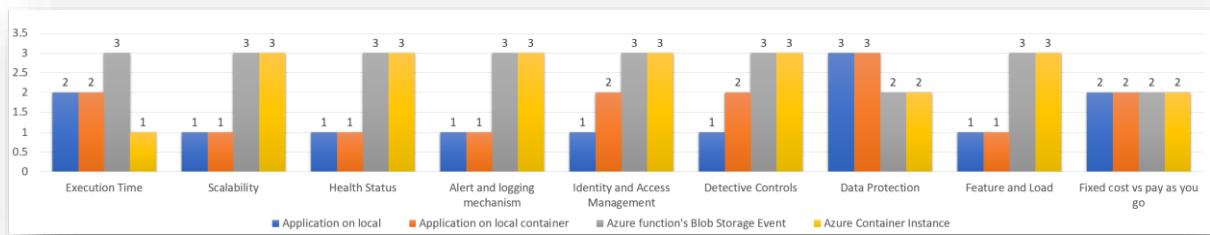
Over the course of a workload's existence, cost optimization is a continuous process that needs refinement and improvement. It is evaluated here not based on actual cost spent to build the application as the system used was local and cloud subscription was student plan. Hence, the comparison has been done on static or dynamic charges involved with each setup (on-premises vs Cloud). Any application running on local, custom designed or third-party licensed software, runs with a fixed cost. Whereas cloud-based solution is pay-as-you-go model where cost is totally based on type of resource provisioned, size of application, compute power etc.

### 6.6 Discussion

The analysis on the SSCAR has been done based-on best industry practises and as defined in well architected frameworks<sup>11</sup> such as execution time, scalability, health status, logging, data protection and identity & access management. Based on the evaluation, each of these platforms

<sup>11</sup> <https://learn.microsoft.com/en-us/azure/architecture/framework/>

has been rated for each of these factors and Figure 9 shows the graph-based presentation of the evaluation parameters.



**Figure 9**

Ratings are between 1-3 with 3 being the best, 1 being the lowest and 2 on average. It is evident that Cloud-based applications performs better in almost every aspect and hence the popularity these days whereas server-based application promises fixed cost but is not efficient and scalable in terms of storage or computing power. The other parameters can still be controlled using some third-party tools. Among cloud-based solutions, Azure functions have a certain degree of flexibility but limited functionality over cloud-based containers as containers need some additional time to pull the image and run applications. In serverless model, security is a shared service model where responsibilities are shared between the customer and cloud provider whereas security is fully managed by the organization and have full control to restrict access and own the infrastructure, in a server-based model. Finally, serverless can occasionally be vulnerable to denial-of-wallet attacks (DoW) (Marin et al., 2022), whereas monolith server-based architecture has an advantage on the same cost in application deployment (so the rating since server-based has always fixed cost and any operation does not raise cost), but it is highly inefficient when it needs to be scaled up or down to control application performance.

## 7 Conclusion and Future Work

This study has discussed various use-cases of serverless computing, containerization, and various ways to ease the development of security services in the cloud. It describes a serverless design patterns that can be used to build Azure serverless applications and services. The objective of this research is to test the baseline security improvements of traditional containerized serverless computing models by implementing RBAC, vulnerability test, and managed identity in Azure. The approach can be implemented when application systems are broken down into little bits and generated smaller sized container images that can be developed in a shorter time frame and put to production, as demonstrated by this experiment. If an application bug is discovered in production, developers will be able to simply roll back the system utilizing the containerized image change management mechanism while maintaining a hierarchical container image. Because of the inherent stability and ease of service management, this architecture is a good approach for businesses to use on a big scale in Azure or any other cloud services.

This work can be further expanded to generate two or more containers with SSCAR utilizing different cloud services and managed in Kubernetes or similar services, which would be an excellent solution to eliminate vendor lock-in for an application.

## References

- Abdelbaky, M., Diaz-Montes, J., Parashar, M., Unuvar, M., & Steinder, M. (2015). Docker Containers across Multiple Clouds and Data Centers. *Proceedings - 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing, UCC 2015*, 368–371. <https://doi.org/10.1109/UCC.2015.58>
- Adzic, G., & Chatley, R. (2017). *Serverless Computing: Economic and Architectural Impact*. 6, pages. <https://doi.org/10.1145/3106237.3117767>
- al Jawarneh, I. M., Bellavista, P., Bosi, F., Foschini, L., Martuscelli, G., Montanari, R., & Palopoli, A. (2019). Container Orchestration Engines: A Thorough Functional and Performance Comparison. *IEEE International Conference on Communications, 2019-May*. <https://doi.org/10.1109/ICC.2019.8762053>
- Ao, L., Izhikevich, L., Voelker, G. M., & Porter, G. (2018). *Sprocket: A Serverless Video Processing Framework*. <https://doi.org/10.1145/3267809.3267815>
- Augustyn, D. R., Wyciślik, Ł., & Sojka, M. (2022). The FaaS-Based Cloud Agnostic Architecture of Medical Services—Polish Case Study. *Applied Sciences 2022, Vol. 12, Page 7954, 12(15)*, 7954. <https://doi.org/10.3390/AP12157954>
- Ayoub Kamal, M., Wahab Raza, H., Alam, M., & Mazliham, M. S. (2020). Highlight the Features of AWS, GCP and Microsoft Azure that Have an Impact when Choosing a Cloud Service Provider Optimal Placement of generator for Power System Losses Reduction in Substation using SSA Algorithm View project Conference Paper View project. *International Journal of Recent Technology and Engineering, 5*, 2277–3878. <https://doi.org/10.35940/ijrte.D8573.018520>
- Bebortta, S., Das, S. K., Kandpal, M., Kumar Barik, R., & Dubey, H. (2020). Geo-Information Geospatial Serverless Computing: Architectures, Tools and Future Directions. <https://doi.org/10.3390/ijgi9050311>. <https://doi.org/10.3390/ijgi9050311>
- Becker Westphall, C., & Olmsted, A. (2016). *CLOUD COMPUTING 2016 The Seventh International Conference on Cloud Computing, GRIDs, and Virtualization Membership certificates View project MAIDAM in IoT/Fog/Cloud-Mutual Authentication, Intrusion Detection and Autonomic Management in IoT/Fog/Cloud Environments View project*. <https://www.researchgate.net/publication/298785531>
- Bermejo, B., & Juiz, C. (2022). A general method for evaluating the overhead when consolidating servers: performance degradation in virtual machines and containers. *The Journal of Supercomputing, 78(9)*, 11345–11372. <https://doi.org/10.1007/s11227-022-04318-5>
- Brady, K., Moon, S., Nguyen, T., & Coffman, J. (2020). Docker Container Security in Cloud Computing; Docker Container Security in Cloud Computing. *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. <https://doi.org/10.1109/CCWC47524.2020.9031195>
- Brasser, F., Jauernig, P., Pustelnik, F., Sadeghi, A.-R., & Stapf, E. (n.d.). *Trusted Container Extensions for Container-based Confidential Computing*.
- Burkat, K., Pawlik, M., Balis, B., Malawski, M., Vahi, K., Rynge, M., da Silva, R. F., & Deelman, E. (2021). Serverless Containers – Rising Viable Approach to Scientific Workflows. *2021 IEEE 17th International Conference on eScience (eScience)*, 40–49. <https://doi.org/10.1109/eScience51609.2021.00014>
- Cheng, X., Lyu, F., Quan, W., Zhou, C., He, H., Shi, W., & Shen, X. (2019). Space/Aerial-Assisted Computing Offloading for IoT Applications: A Learning-Based Approach. *IEEE Journal on Selected Areas in Communications, 37(5)*, 1117–1129. <https://doi.org/10.1109/JSAC.2019.2906789>
- Expósito Jiménez, V. J., Zeiner, H., & Juan Expósito Jiménez, V. (2018). *Serverless Cloud Computing : A Comparison Between “Function as a Service” Platforms Robot technology for the wood industry (RobWood) View project Secure Contactless Sphere. Smart RFID-Technologies for a Connected World View project SERVERLESS CLOUD COMPUTING: A COMPARISON BETWEEN “FUNCTION AS A SERVICE” PLATFORMS*. 15–22. <https://doi.org/10.5121/csit.2018.80702>
- Hassan, H. B., Barakat, S. A., & Sarhan, Q. I. (2021). Survey on serverless computing. *Journal of Cloud Computing, 10(1)*, 39. <https://doi.org/10.1186/s13677-021-00253-7>
- Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C.-C., Khandelwal, A., Pu, Q., Shankar, V., Carreira, J., Krauth, K., Yadwadkar, N., Gonzalez, J. E., Popa, R. A., & Patterson, D. A. (2019). *Cloud Programming Simplified: A Berkeley View on Serverless Computing*.

- Khan, A. (2017). Key Characteristics of a Container Orchestration Platform to Enable a Modern Application. *IEEE Cloud Computing*, 4(5), 42–48. <https://doi.org/10.1109/MCC.2017.4250933>
- Kratzke, N. (2014). Lightweight Virtualization Cluster How to Overcome Cloud Vendor Lock-In. *Journal of Computer and Communications*, 02(12), 1–7. <https://doi.org/10.4236/JCC.2014.212001>
- Kwon, S., & Lee, J. H. (2020). DIVDS: Docker Image Vulnerability Diagnostic System. *IEEE Access*, 8, 42666–42673. <https://doi.org/10.1109/ACCESS.2020.2976874>
- Leitner, P., Wittern, E., Spillner, J., & Hummer, W. (2019). A mixed-method empirical study of Function-as-a-Service software development in industrial practice. *Journal of Systems and Software*, 149, 340–359. <https://doi.org/10.1016/J.JSS.2018.12.013>
- Lenarduzzi, V., & Panichella, A. (2021). Serverless Testing: Tool Vendors' and Experts' Points of View. *IEEE Software*, 38(1), 54–60. <https://doi.org/10.1109/MS.2020.3030803>
- Marin, E., Perino, D., & di Pietro, R. (2022). Serverless computing: a security perspective. *Journal of Cloud Computing*, 11, 69. <https://doi.org/10.1186/s13677-022-00347-w>
- Morenets, I., & Shabinskiy, A. (2020). Serverless Event-driven Applications Development Tools and Techniques. *NaUKMA Research Papers. Computer Science*, 3(0), 36–41. <https://doi.org/10.18523/2617-3808.2020.3.36-41>
- Pahl, C. (2015). *Containerization and the PaaS Cloud; Containerization and the PaaS Cloud*. <https://doi.org/10.1109/MCC.2015.51>
- Pérez, A., Moltó, G., Caballer, M., & Calatrava, A. (2018). Serverless computing for container-based architectures. *Future Generation Computer Systems*, 83, 50–59. <https://doi.org/10.1016/J.FUTURE.2018.01.022>
- Pogiatzis, A., & Samakovitis, G. (2021). An Event-Driven Serverless ETL Pipeline on AWS. *Applied Sciences*, 11(1). <https://doi.org/10.3390/app11010191>
- Scheepers, T. (2014). *Virtualization and Containerization of Application Infrastructure: A Comparison*.
- Shu, R., Gu, X., & Enck, W. (n.d.). A Study of Security Vulnerabilities on Docker Hub. *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. <https://doi.org/10.1145/3029806>
- Simonsson, J., Zhang, L., Morin, B., Baudry, B., & Monperrus, M. (2021). Observability and chaos engineering on system calls for containerized applications in Docker. *Future Generation Computer Systems*, 122, 117–129. <https://doi.org/10.1016/J.FUTURE.2021.04.001>
- Vasanthakumari, N., & Arulmurugan, R. (2022). *Reorganizing Virtual Machines as Docker Containers for Efficient Data Centres* (pp. 201–211). [https://doi.org/10.1007/978-3-030-78750-9\\_14](https://doi.org/10.1007/978-3-030-78750-9_14)
- Watada, J., Roy, A., Kadikar, R., Pham, H., & Xu, B. (2019). Emerging Trends, Techniques and Open Issues of Containerization: A Review. *IEEE Access*, 7, 152443–152472. <https://doi.org/10.1109/ACCESS.2019.2945930>
- Wen, J. (2022). *A Literature Review on Serverless Computing; A Literature Review on Serverless Computing*. <https://doi.org/https://doi.org/10.48550/arXiv.2206.12275>