Darshan Chetty
St10226392
CLLDV POE Part 1

A: Contrast how traditional and cloud applications differ by examining the following table

Traditional On-Premises		Modern Cloud	
On-premises definition	**On-premises example**	**Cloud definition**	**Cloud example**
Monolithic: Monolithic architecture refers to a software program that was developed as a single, standalone, integrated system. A software program with a monolithic architecture is one that was developed as a single, cohesive, and autonomous system.	Monolithic: SaaS software encompasses a variety of components, such as a web server, load balancer, a feature for displaying product images from a catalogue, a platform for placing orders, a payment processing feature, and a delivery mechanism.	Decomposed: This approach scatters data across different cloud-based storage systems instead of keeping it concentrated in a single cloud location. Think of BitTorrents, where data is distributed to devices connected to the torrent network – it's a form of cloud storage that's not reliant on a central hub.	Decomposed: Unlike centralized systems that rely on the organization's resources for scalability, this design offers almost limitless scalability. Blockchain serves as another prime example of a decentralized system, highlighting this distinctive approach to

			virtually boundless growth.
Designed for predictable scalability: Sufficient performance is crucial for ensuring that software or applications function effectively. Cloud scalability is the solution employed to manage the increasing demands. Scalability comes into play when there's a need to continuously allocate additional resources to handle the growing workload in a stable manner.	Designed for predictable scalability: Imagine you're the owner of a company, and your database started small but has grown with your business's expansion. Now, all you have to do is ask your cloud service provider to expand your database's capacity to meet increased demand. Scalability is employed to fulfill fixed needs, and, like other cloud services, it involves clients paying for each usage. To sum it up, scalability proves beneficial when dealing with consistent and substantial workloads.	Designed for elastic scale: Elasticity in the context of cloud computing refers to the cloud's capability to quickly adjust and scale infrastructure resources in response to sudden fluctuations in demand. This adaptability enhances cost efficiency by only using the necessary resources when needed. It's not applicable in all situations, but it shines when resource demands experience rapid and temporary spikes and drops.	Designed for elastic scale: To efficiently handle the surge in online shopping activity during holidays like Christmas, it's beneficial to utilize cloud elasticity services instead of traditional cloud scalability. This means we can easily boost our resources when needed and, once the holiday season passes, effortlessly scale them down or remove them as required.
Relational database: A relational database, a specific type of database, stores and provides access to interconnected data points. The foundation of relational databases is the relational model, which is	Relational database: Here's a simplified illustration of two tables that a small business might use to handle product orders. The first table, containing customer information, holds details like names,	Polyglot persistence (mix of storage technologies): Polyglot persistence is a concept in the realm of enterprise data management where experts choose to utilize multiple data storage technologies to cater to	Polyglot persistence (mix of storage technologies): An online store serves as a great example of how a company can effectively utilize polyglot persistence. In the context of an online store,

a straightforward method of organizing data into tables. Within a relational database, every row in a table represents a unique record, identified by a special ID called a key. Since each record typically has values for each property listed in the table's columns, it's easy to establish how data points relate to one another.	addresses, phone numbers, shipping, and billing info, with each piece of data neatly organized into separate columns. Each row in this table is assigned a unique ID (a key). The second table is specifically for customer orders, and it includes entries for the customer ID, the product ordered, quantity, size, colour choices, and so forth. Importantly, it doesn't include the customer's personal details like their name or contact information. The key commonality between these two tables is the ID column. However, thanks to this shared field, a relational database can establish a connection between them. When the company's order processing system submits an order to the database, it can access the customer order table, extract the relevant product order details, and use the customer ID from that table to locate the customer's billing and shipping information in the customer info table. This allows the company to efficiently retrieve the correct item from their warehouse, ensure timely delivery of the customer's order, and promptly receive payment, all thanks to this interconnected database setup.	different types of data and their specific storage needs. The core notion here is that an application can make use of several primary databases simultaneously.	various types of data are employed just for the shopping cart function, including transactional records, session data, inventory details, completed orders, and customer profiles. In the past, businesses used a single database system to handle all the data needed for their e-commerce applications. This approach often required extensive data conversions to fit multiple data formats into a single relational database. However, according to the principles of polyglot persistence, it's recommended to separate shopping cart and e-commerce data into databases that are best suited for each specific type of data.
Synchronized processing: Creating reactive systems is most effectively done through synchronous architecture, often	Synchronized processing: To grasp how synchronous programming functions, think of a telephone	Asynchronous processing: Asynchronous programming, particularly well-suited	Asynchronous processing: Asynchronous communication methods, like texting, provide

referred to as a blocking architecture. This approach, similar to a single-threaded model, adheres to a precise sequence of actions, allowing each task to be executed one by one and in a specific order. While one task is in progress, it holds up the instructions for subsequent tasks. It's like a relay, where each operation passes the baton to the next one in line.	conversation. In this scenario, when one person is talking on the phone, the other person usually responds promptly as soon as the first person has finished speaking.	for networking and communications, is a multi-threaded approach. In contrast to blocking designs, asynchronous architecture doesn't hold up subsequent operations while one or more tasks are in progress. In asynchronous programming, multiple interconnected tasks can run simultaneously without waiting for others to finish. Asynchronous communicators don't respond immediately upon receiving a message; instead, they choose a convenient or practical time to read and process it.	flexibility. When someone sends a text message, the recipient can respond at their convenience. Meanwhile, the sender is free to engage in other activities while awaiting a reply.
Design to avoid failures (MTBF): The term MTBF (mean time between failures) represents the average period between repairable failures of a technical product. This metric is employed to monitor the product's availability and dependability. Essentially, the longer the time between failures, the higher the system's reliability.	Design to avoid failures (MTBF): Manufacturers find the mean time between failures (MTBF) a valuable measure of reliability that can be applied at different stages of product development and production. Nowadays, it's commonly used in designing mechanical and electrical systems, ensuring safe industrial operations, making purchasing decisions for materials, and more.	Design for failure (MTTR): The mean time to repair, known as MTTR, represents the average duration required to fix a system, typically of a technical or mechanical nature. It encompasses both the time spent on repairs and any testing required. This timer keeps running until the system is completely back in working order.	Design for failure (MTTR): Imagine most of the water heaters in your building have thermostat issues, which are relatively straightforward and affordable for most people to fix. However, one stands out from the rest. This particular unit, due to strange noises and mineral buildup, requires more extensive repairs and maintenance to prevent a potentially costly leak or explosion. Consequently, the time it takes to fix this thermostat will be considerably longer, making the average thermostat repair time appear unusually lengthy.
Occasional large updates: A software upgrade isn't just an addition to your current application; it's an entirely new iteration of the software product. Consequently, the outdated application is	Occasional large updates: When dealing with games like World of Warcraft that have an active player base, it's beneficial to consider implementing gradual updates. These updates can introduce	Frequent small updates: Software updates provide programmers with the opportunity to enhance a product, striving for its utmost excellence. They offer incremental, periodic enhancements	Frequent small updates: In games like Terraria, where players engage periodically, larger updates are favoured. These updates, occurring every couple of months, provide players with a

substituted with this upgraded version. Upgrades are deployed to bring significant improvements and enhancements to the existing application. This can include a completely revamped user interface, a host of exciting new features, or substantial structural alterations.	small, fresh content elements at regular intervals, enhancing the game's overall vitality. Moreover, this approach avoids the necessity for large downloads or abrupt changes in gameplay, ensuring a smoother experience for gamers.	rather than sweeping alterations. Unlike software upgrades, updates build upon your existing software. Automatic background updates are occasionally conducted, often being crucial for the ongoing smooth operation of your product. This necessity arises from the fact that software updates encompass support for new drivers and hardware, along with the resolution of newly discovered issues and security vulnerabilities. The software's functionality is refined and enhanced, though not fundamentally overhauled.	substantial amount of new content when they return to the game. The aim is to rekindle the excitement they initially found in the game. Additionally, returning players often only have one question in mind.
Manual management: When software necessitates manual updates and precise placement in the correct directories to function, any mistakes in this process can lead to issues.	Manual management: Upon downloading a game, you might encounter a situation where the README file provides installation instructions that require you to navigate to specific directories, create folders, and place files in precise locations to ensure the software functions correctly.	Automated self-management: When software has the capability to autonomously organize files into the correct folders, eliminating the necessity for manual configuration.	Automated self-management: You can obtain, install, or update your software without the need to manually organize files and directories.
Snowflake servers: A snowflake server is a server whose current configuration differs significantly from the ideal setup. These servers can pose substantial challenges, and sometimes, when you wish to replace such a server, it's not always clear what specific elements are essential for its proper functioning. As a result, the newly launched server may struggle to handle the workload as expected.	Snowflake servers: Imagine you've developed a web server and deployed it on an existing and adaptable infrastructure. Users keep sending requests to the server, and over time, you realize that modifications are needed. The advantage here is that you don't need to migrate your existing data to a new machine. After making necessary adjustments to the	Immutable infrastructure: Immutable infrastructure pertains to IT server infrastructure that remains unaltered once it's been deployed. This concept is often linked to software engineering methodologies like DevOps and continuous delivery. If any adjustments or enhancements are required, an entirely new, updated instance is deployed onto the server. Setting up fresh configurations in the	Immutable infrastructure: Think of Docker as an example of immutable infrastructure. In an immutable image, you have everything needed to run the application, including the source code and all essential components. Docker containers follow a core principle: once you create an image, you can't modify it. In simpler terms, any changes result in a completely new image.

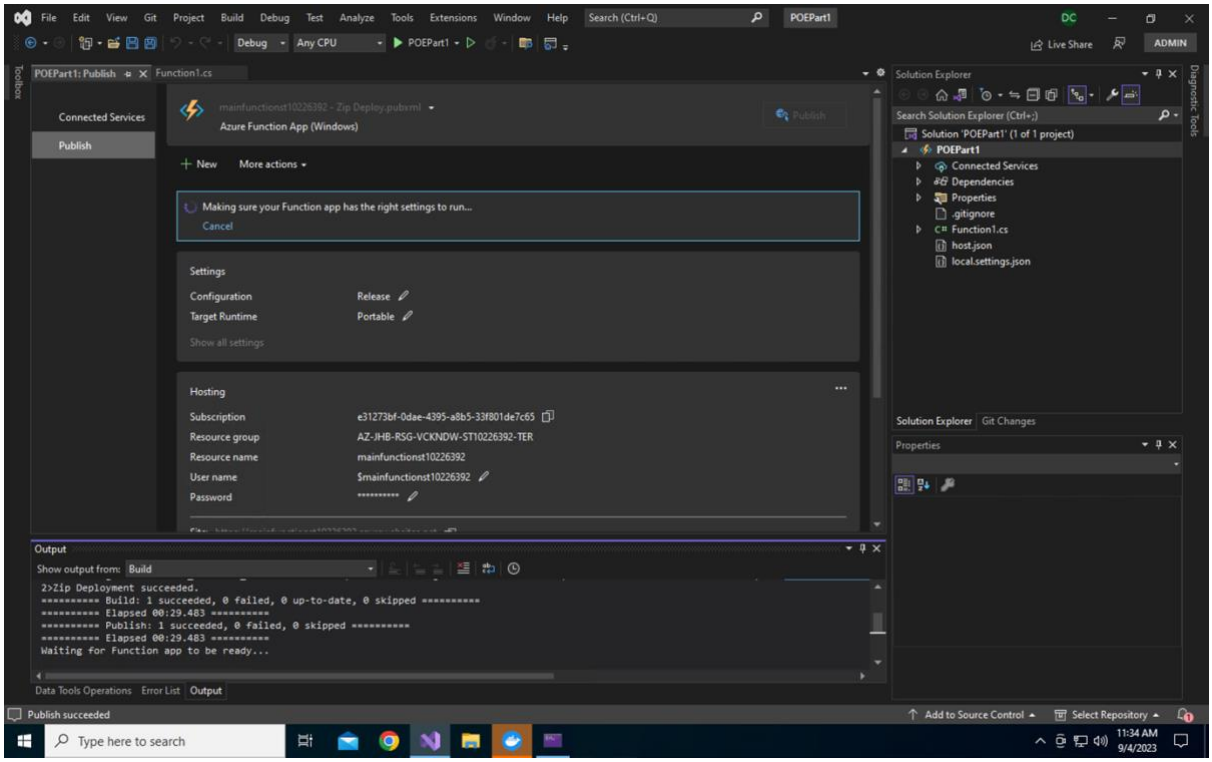
	current version, you can smoothly implement the upgrade. However, what if your upgrade doesn't go as smoothly as expected? Most enhancements and changes encounter some hiccups, right? What if you inadvertently compromise your system's security or end up with an unintended version of the upgrade? Making changes, facing challenges along the way, and receiving a product that's not entirely understood come with inherent risks. Testing and validating a product becomes challenging when you're dealing with the unknown. This means you might find yourself dealing with complex and hard-to-resolve issues.	cloud can be accomplished swiftly, typically in just a few minutes.	

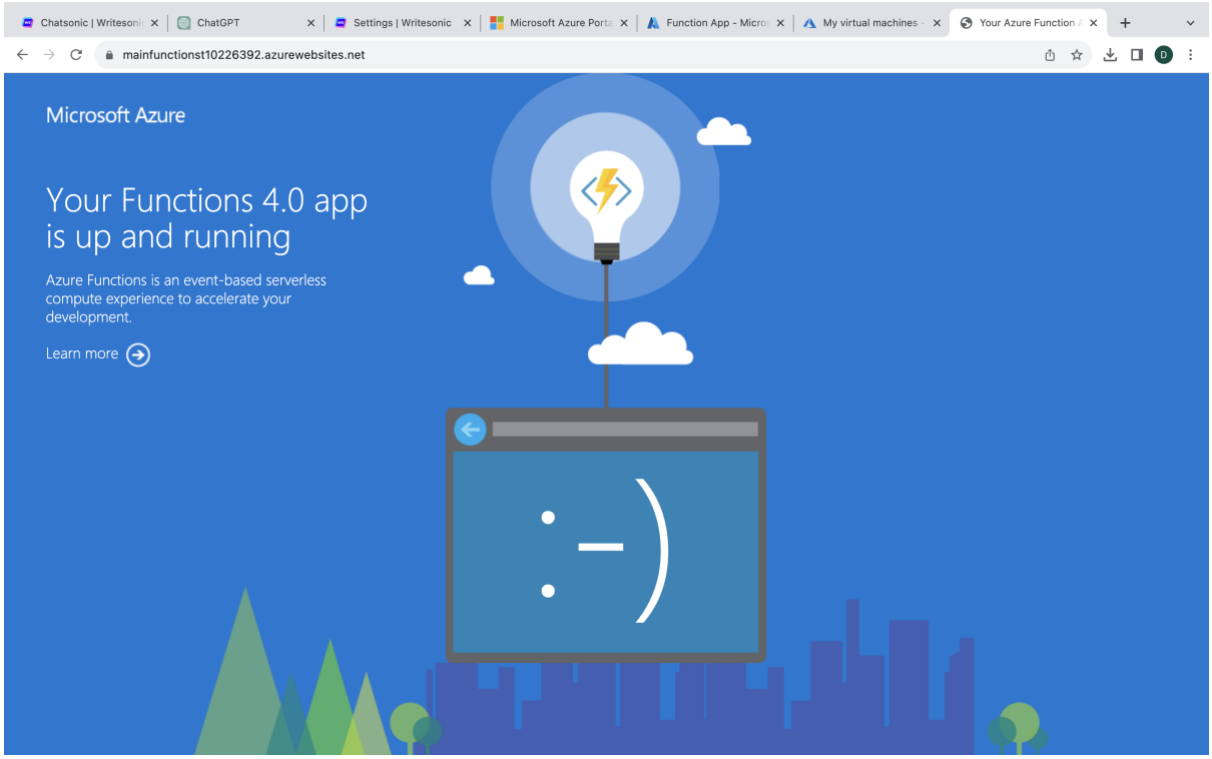
B. Deploy an Azure Function compute service to the cloud

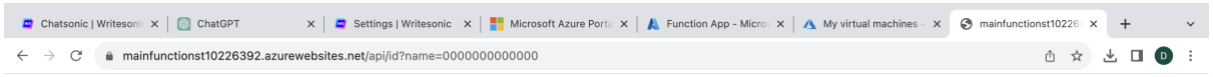
```
using System;
using System.IO;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;

namespace POEPart1
{

    public static class Function1
    {
        [FunctionName("St10226392")]
        public static async Task<IActionResult> Run(
    [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = "id")] HttpRequest req,
    ILogger log)
```

```
    {
        log.LogInformation("C# HTTP trigger function processed a request.");

        string[] ids = { "0000000000000", "7272727272727", "3986533009955" };
        string[] names = { "Ed", "Jake", "Phil" };
        string[] centres = { "Dischem", "Alpha Pharmacy", "Momentum" };
        string[] vaccines = { "Pfizer", "J&J", "Pfizer" };
        string[] vaccinationStatus = { "Fully Vaccinated", "Partially Vaccinated", "Not Vaccinated" };

        string name = req.Query["name"];
        string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
        dynamic data = JsonConvert.DeserializeObject(requestBody);

        name = name ?? data?.name;

        int index = Array.FindIndex(ids, id => id.Equals(name, StringComparison.OrdinalIgnoreCase));
        string active = "ID not Found.";


        if (index != -1)
        {
            active =
                "\n-------------------------------------------------" +
                "\nVaccination Information" +
                "\n-------------------------------------------------" +
                "\nID: " + ids[index] +
                "\nName: " + names[index] +
                "\nVaccination Centre: " + centres[index] +
                "\nVaccine Name: " + vaccines[index] +
                "\nVaccination Status: " + vaccinationStatus[index] +
                "\n" + "-------------------------------------------------";


        }

        string responseMessage = string.IsNullOrEmpty(name)
            ? $"{active}"
            : $"{active}\n\nThis HTTP triggered function executed successfully.";

        return new OkObjectResult(responseMessage);
    }


}
}
```

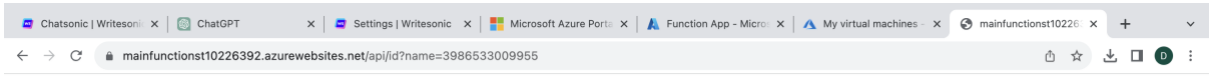
SCREENSHOTS OF FUNCTION WORKING IN BROWSER:


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
POEPart1
Debug
Any CPU
POEPart1
Live Share
ADMIN
POEPart1: Publish
Function1.cs
Connected Services
Publish
Azure Function App (Windows)
New
More actions
Making sure your Function app has the right settings to run...
Cancel
Settings
Configuration
Release
Target Runtime
Portable
Show all settings
Hosting
Subscription
e31273bf-0dae-4395-a8b5-33f801de7c65
Resource group
AZ-JHB-RSG-VCKNDW-ST10226392-TER
Resource name
mainfunctionst10226392
User name
$mainfunctionst10226392
Password
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'POEPart1' (1 of 1 project)
POEPart1
Connected Services
Dependencies
Properties
.gitignore
Function1.cs
host.json
local.settings.json
Solution Explorer
Git Changes
Properties
Output
Show output from: Build
2>Zip Deployment succeeded.
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Elapsed 00:29.483 ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
========== Elapsed 00:29.483 ==========
Waiting for Function app to be ready...
Data Tools Operations
Error List
Output
Publish succeeded
Add to Source Control
Select Repository
Type here to search
11:34 AM
9/4/2023

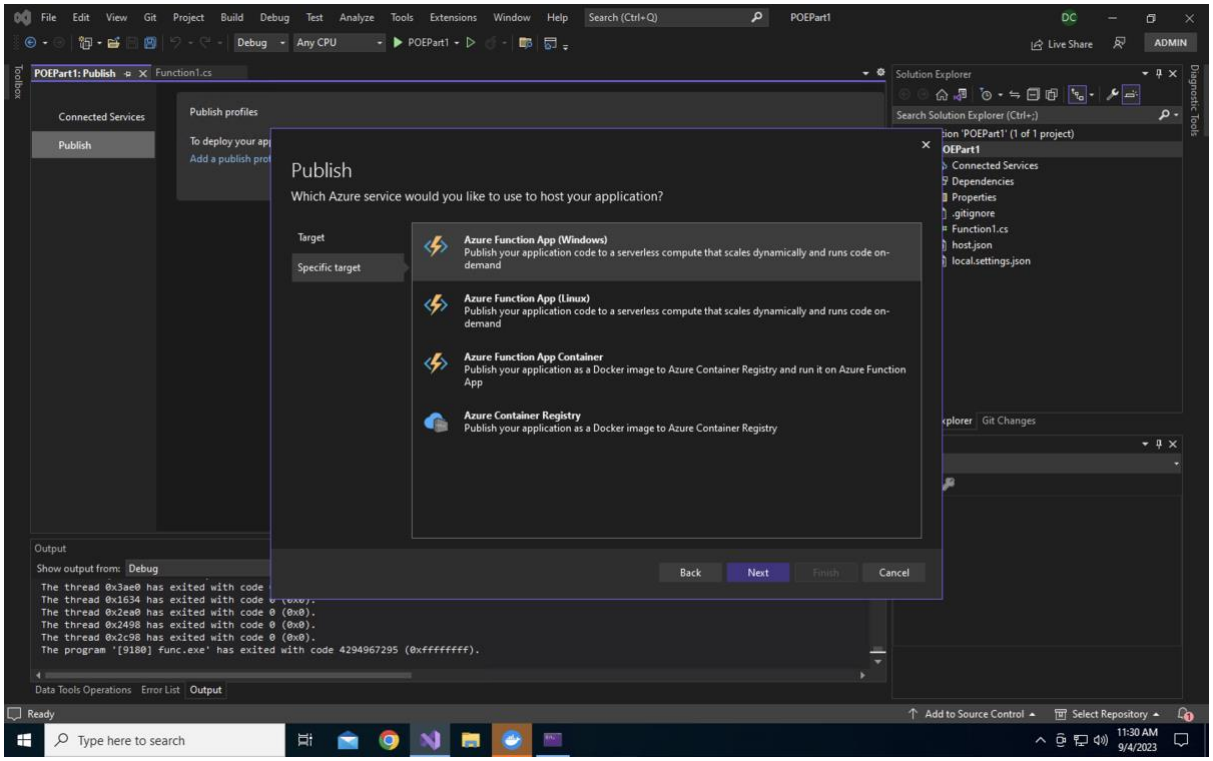

Chatsonic | Writesonic
ChatGPT
Settings | Writesonic
Microsoft Azure Porta
Function App - Micro
My virtual machines
Your Azure Function
mainfunctionst10226392.azurewebsites.net
Microsoft Azure
Your Functions 4.0 app is up and running
Azure Functions is an event-based serverless compute experience to accelerate your development.
Learn more

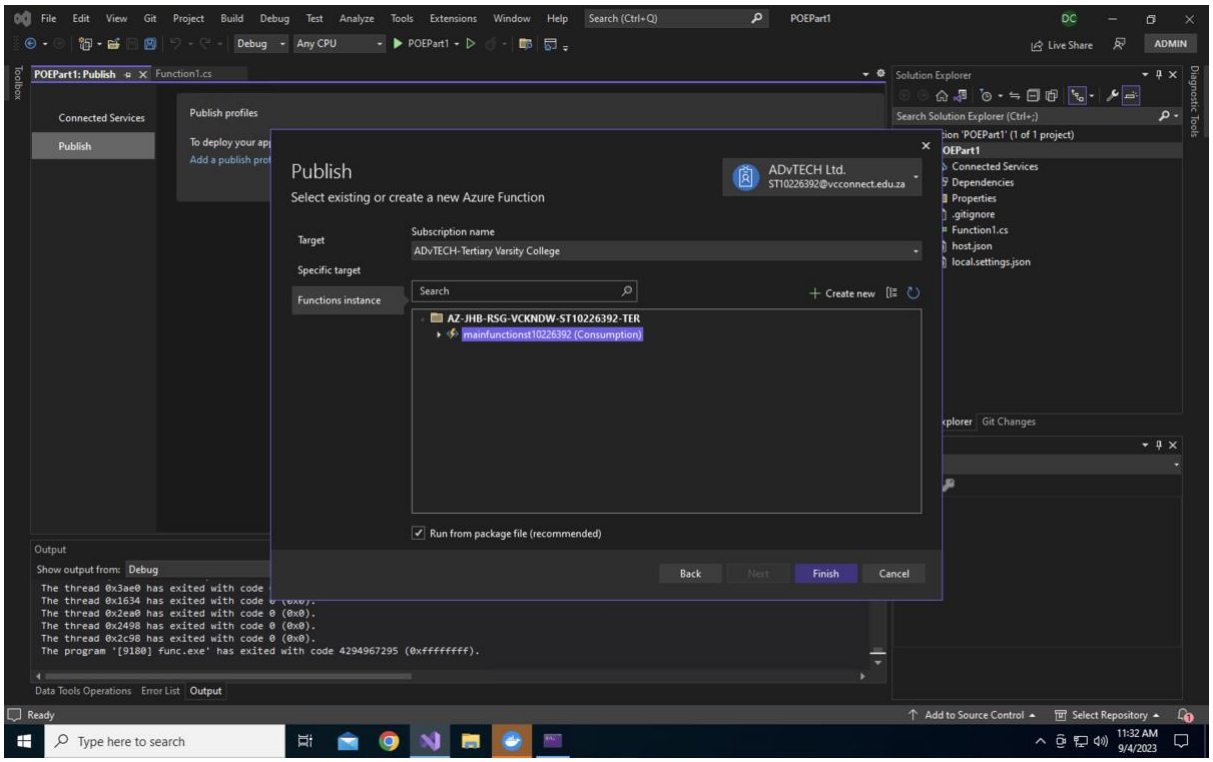


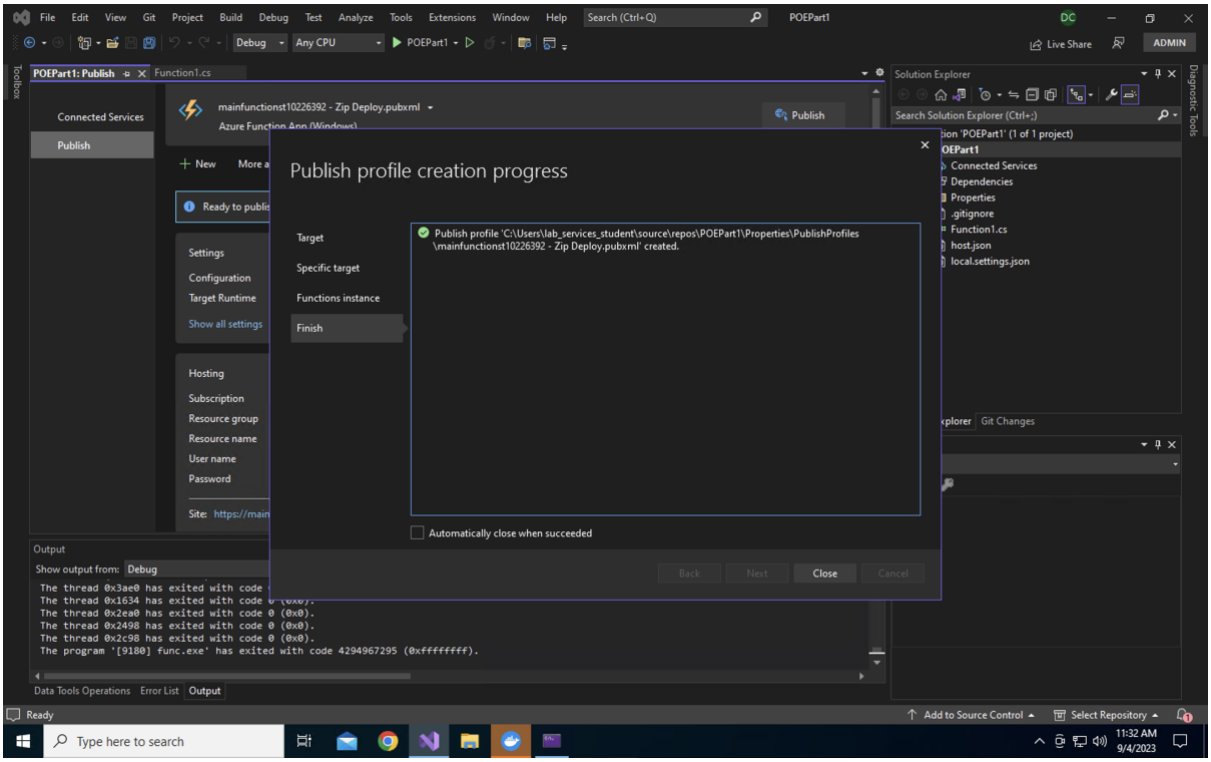

```
---------------------------------------------------
Vaccination Information
---------------------------------------------------
ID: 0000000000000
Name: Ed
Vaccination Centre: Dischem
Vaccine Name: Pfizer
Vaccination Status: Fully Vaccinated
---------------------------------------------------

This HTTP triggered function executed successfully.
```




```
---------------------------------------------------
Vaccination Information
---------------------------------------------------
ID: 3986533009955
Name: Phil
Vaccination Centre: Momentum
Vaccine Name: Pfizer
Vaccination Status: Not Vaccinated
---------------------------------------------------

This HTTP triggered function executed successfully.
```


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
POEPart1
Debug
Any CPU
POEPart1
Live Share
ADMIN
POEPart1: Publish
Function1.cs
Connected Services
Publish
Publish profiles
Publish
Which Azure service would you like to use to host your application?
Target
Specific target
Azure Function App (Windows)
Publish your application code to a serverless compute that scales dynamically and runs code on-demand
Azure Function App (Linux)
Publish your application code to a serverless compute that scales dynamically and runs code on-demand
Azure Function App Container
Publish your application as a Docker image to Azure Container Registry and run it on Azure Function App
Azure Container Registry
Publish your application as a Docker image to Azure Container Registry
Back
Next
Finish
Cancel
Solution Explorer
Search Solution Explorer (Ctrl+;)
Connected Services
Dependencies
Properties
.gitignore
Function1.cs
host.json
local.settings.json
Git Changes
Output
Show output from: Debug
The thread 0x3ae0 has exited with code
The thread 0x1634 has exited with code
The thread 0x2ea0 has exited with code 0 (0x0).
The thread 0x2498 has exited with code 0 (0x0).
The thread 0x2c98 has exited with code 0 (0x0).
The program '[9180] func.exe' has exited with code 4294967295 (0xffffffff).
Data Tools Operations
Error List
Output
Ready
Add to Source Control
Select Repository
Type here to search
11:30 AM
9/4/2023


Publish
Select existing or create a new Azure Function
ADvTECH Ltd.
ST10226392@vcconnect.edu.za
Target
Specific target
Functions instance
Subscription name
ADvTECH-Tertiary Varsity College
Search
Create new
AZ-JHB-RSG-VCKNDW-ST10226392-TER
mainfunctionst10226392 (Consumption)
Run from package file (recommended)
Back
Next
Finish
Cancel


Publish profile creation progress
Target
Specific target
Functions instance
Finish
Publish profile 'C:\Users\lab_services_student\source\repos\POEPart1\Properties\PublishProfiles\mainfunctionst10226392 - Zip Deploy.pubxml' created.
Automatically close when succeeded
Back
Next
Close
Cancel

Link:
https://mainfunctionst10226392.azurewebsites.net/api/id?name=3986533009955

REFERENCES:

Bevans, D., 2022. Asynchronous vs. Synchronous Programming. [online] Mendix. Available at: <https://www.mendix.com/blog/asynchronous-vs-synchronous-programming/> [Accessed 28 August 2022].

Bevans, D., 2023. *Asynchronous vs. Synchronous Programming: Key Similarities and Differences.* [Online]
Available at: https://www.mendix.com/blog/asynchronous-vs-synchronous-programming/#:~:text=Sync%20is%20single%2Dthread%2C%20so,be%20answered%20by%20the%20server.

Brunskill, V.-L., 2023. *polyglot persistence.* [Online]
Available at: https://www.techtarget.com/searchapparchitecture/definition/polyglot-persistence#:~:text=Polyglot%20persistence%20is%20a%20conceptual,that%20live%20within%20enterprise%20applications.

Cook, B., 2012. *Which approach is better: Frequent small updates, or occasional large ones?.* [Online]
Available at: https://gamedev.stackexchange.com/questions/25167/which-approach-is-better-frequent-small-updates-or-occasional-large-ones

Harris, C., 2023. *Microservices vs. monolithic architecture.* [Online]
Available at: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith#:~:text=A%20monolithic%20architecture%20is%20a%20singular%2C%20large%20computing%20network%20with,of%20the%20service%2Dside%20interface.

Microsoft, 2023. *Design applications for scaling.* [Online]
Available at: https://learn.microsoft.com/en-us/azure/well-architected/scalability/design-scale

Neal Ford, M. R. P. S. Z. D., 2023. *Chapter 4. Architectural Decomposition.* [Online]
Available at: https://www.oreilly.com/library/view/software-architecture-the/9781492086888/ch04.html

Oracle, 2023. *What is a relational database.* [Online]
Available at: https://www.oracle.com/za/database/what-is-a-relational-database/

Rockwell Automation, 2023. *Mean time between failures.* [Online]
Available at: https://fiixsoftware.com/maintenance-metrics/mean-time-between-fail-maintenance/#:~:text=MTBF%20is%20used%20to%20anticipate,help%20you%20avoid%20costly%20breakdowns.

Smirnoff, P., 2017. *Manual vs. Automated Key Management.* [Online]
Available at: https://www.cryptomathic.com/news-events/blog/manual-vs.-automated-key-management

Sumo Logic, 2023. *Mutable and immutable infrastructure - definition & overview.* [Online] Available at: https://www.sumologic.com/glossary/mutable-and-immutable-infrastructure/#:~:text=The%20benefits%20of%20an%20immutable,configuration%20drift%20and%20snowflake%20servers.