# Cloud Computing - Mini Project Report
# Breaking Down Monoliths
# April 2023

Submitted By:

Name :

SRN : PES1UG20CS573

PES1UG20CS584

PES1UG20CS

PES1UG20CS608

VI Semester Section J

PES University

# Short  Description and Scope of the Project

The monolithic architecture is a traditional software architecture that consists of a single, large application that is developed, deployed, and maintained as a single unit. However, this architecture can become difficult to manage as the application grows in size and complexity. Microservices architecture, on the other hand, breaks down the application into smaller, independent services that can be developed, deployed, and scaled independently. This architecture can help to improve the maintainability, scalability, and reliability of the system

In this project, the focus is on breaking down a monolithic arithmetic operation application into microservices. The application can be decomposed into several microservices, each responsible for a specific arithmetic operation, such as addition, subtraction, multiplication, and division. The project involves identifying the data each microservice will need to perform its function, defining the API for each microservice, developing and testing each microservice, and integrating the microservices into the larger system.

The scope of this project is to break down a monolithic architecture that consists of arithmetic operations into microservices. The project involves decomposing the application into smaller, loosely-coupled services that can be developed, deployed, and scaled independently. The main focus of the project is to identify the core functionality of the application, design the API for each microservice, develop and test each microservice, and integrate the microservices into the larger system.

# Methodology

## Task 0

We have added the required modules to the requirements.txt file of the landing-service and wrote a Dockerfile for the application. The landing-service requires Flask, Flask-Restful, and Requests modules to run. The Dockerfile uses the python:3.8-alpine image and copies the contents of the requirements.txt file to /app/requirements.txt. The working directory should is set to /app, and the Dockerfile runs the command to install the required modules using pip and apk package manager. Finally, the contents of the ./app folder is copied to /app and the command to run the Python application is executed.

## Task 1

To fix the variable type of incoming values, we convert them from string to float data types. This is done using the float() function in Python. We have modified the code in the landing service app to include this conversion.

To handle the None type exception, we have added an error handling code to the landing service app. We used a try-except block to catch any exceptions that are raised when no values are provided by index.html. In the except block, we set default values for the variables and continue with the arithmetic operations. This will prevent the app from crashing when no values are provided.

**Task 2**

The methodology can be as follows:

- Identifying the functionalities: The functionalities that can be decoupled from the monolithic application are the four mathematical operations - Addition, subtraction, multiplication, and division.
- Develop microservices: Developed each mathematical operation as a separate microservice using Flask framework.
- Test microservices: Tested each microservice using Postman to ensure that all the functions work correctly and meet the requirements.
- Containerize microservices: we Containerized each microservice using Docker, ensuring that all the necessary dependencies are included.
- Orchestrate microservices: Used an orchestration tool, Docker Compose to manage the microservices and route traffic between them.
- Integrate microservices: Integrated the microservices into the main application, replacing the corresponding functionality in the monolithic application with calls to the microservices.

## Results and Conclusions

The result of breaking down a monolithic arithmetic application to microservices depends on the complexity and size of the application, as well as the benefits and drawbacks of each architecture. Here are some possible outcomes:

- Improved scalability and reliability: Microservices can scale independently and handle failures gracefully, while monoliths can be affected by a single point of failure or bottleneck.
- Increased flexibility and maintainability: Microservices can be developed, deployed, and updated independently using different technologies and languages, while monoliths require updating the entire stack and coordinating across teams.
- Reduced development speed and simplicity: Microservices introduce more complexity and overhead in terms of communication, coordination, testing, monitoring, and deployment, while monoliths are easier to develop and deploy with a single code base.
- Changed data management: Microservices use separate databases or schemas for each service, which can improve performance and isolation, but also introduce challenges in data consistency and integrity, while monoliths use a shared database that can ensure transactional consistency but also create coupling and contention.

Breaking down a monolithic arithmetic application into microservices has several benefits. Firstly, it allows for easier maintenance and scalability of the application. Each microservice can be developed and deployed independently, making it easier to manage updates and fixes. Secondly, it allows for greater flexibility in terms of technology choice. Different microservices can use different programming languages or frameworks, depending on the requirements of that specific service. Finally, it enables better fault isolation and fault tolerance. If a microservice fails, it will only affect that specific service and not the entire application.