SIT314/SIT729 – Week 10 Group Activity  
IoT Application deployment using Containers

short line

# Overview

# This activity is to think about the use of Containers in application deployment.

# Tasks

For the same IoT health application that we defined last week, which contains:

* Client that collects health data
* API that allows data to be uploaded from the client and store in a MongoDB Database.
* A MongoDB Database.
* A microservice which calculates the average heartrate of a user.
* A microservice which calculates the yearly fitness of a user.
* A microservice that can provide the fitness level for all the runners in a marathon.
* A microservice which will detect when individuals are having health issues.

Think about the use of containers to structure and implement the deployment of IoT applications.

1. What parts would make most sense to be containerised?

The following components of the IoT health application make the most sense to be containerised:  
  
• API Gateway: Containerising the API that allows data upload from the client enables scalable and independent deployment.  
• MongoDB Database: Running the database in a container helps isolate it and simplifies testing and development.  
• Client Application: The client that collects health data can be containerised to ensure consistency across environments.  
• Microservices:  
 – Average Heartrate Microservice  
 – Yearly Fitness Microservice  
 – Marathon Fitness Microservice  
 – Health Alert Microservice  
  
These microservices should be individually containerised so that each one can be deployed, scaled, and updated independently.

1. What base image would you base each container on?

The appropriate base images for each component are as follows:  
  
• API Server: Use a Node.js or Python (e.g., python:3.11-slim or node:20-alpine) base image depending on backend language.  
• MongoDB Database: Use the official MongoDB Docker image (e.g., mongo:7).  
• Client Application: If it is a web client, use nginx:alpine to serve static files or node for a React app.  
• Microservices:  
 – For microservices written in Python: python:3.11-slim.  
 – For microservices written in Node.js: node:20-alpine.  
 – For performance-critical services, consider using golang or rust base images.

1. Which containers would you autoscale?

The following containers should be set up for autoscaling:  
  
• API Server: To handle varying client requests and traffic patterns.  
• Average Heartrate Microservice: As it may receive high-frequency data processing requests.  
• Health Alert Microservice: Must scale quickly to process real-time health monitoring data.  
• Marathon Fitness Microservice: Likely to be used during marathon events with high request volumes.  
  
The MongoDB container and the client container are less likely to need autoscaling but should be monitored for performance.