Architecture notebook

DPI IoTa Improvements Project

ITC303 - 202330

Group 8 – Clare Strik, Jack Footner, and Sam York

# 1. Purpose

# This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation.

# 2. Architectural goals and philosophy

# The IoTa system has been designed with the philosophy of modularity and scalability in mind. It is built using a microservice architecture, where each service is responsible for a specific task and communicates with other services through a messaging system.

# The architecture goals of IoTa include flexibility, fault-tolerance, and ease of maintenance. By using a modular approach, new services can be added, or existing services can be updated without impacting other services. This also allows for fault-tolerance, as if one service fails, it does not affect the overall system.

# IoTa utilizes containerization through Docker, which provides a way to package each service along with its dependencies, making it easy to deploy and manage. The use of Kubernetes as the container orchestration system allows for efficient scaling, monitoring, and management of the containerized services.

# Additionally, IoTa has a web-based management application for device management, allowing users to easily manage physical and logical devices, as well as mappings between them. The application provides a user-friendly interface for device management, with features such as filtering and multi-select.

# 3. Assumptions and dependencies

## Technical assumptions and dependencies

* The availability of Docker and Docker Compose to manage the deployment of the different components of the system.
* The availability of Kubernetes to enable container orchestration and management of the different services of the system.
* The use of PostgreSQL as the main database management system to store the metadata, physical device data, and mapping between physical and logical devices.
* The use of RabbitMQ as the message broker to facilitate communication between the different services of the system.
* The use of the MQTT protocol to enable the communication between the IoT devices and the system.
* The availability of Python as the main programming language for the different components of the system.
* The use of asynchronous programming and event-driven architecture to enable scalability and performance.
* The project assumes that the necessary infrastructure and resources (such as servers, storage, and networking) are available to deploy and run the Kubernetes cluster.

## Other project assumptions and dependencies

* The availability of skills within the project team in Kubernetes, Python, project management, time management, and general software development.
* The availability of the team resources to implement the changes and meet project deadlines

# 4. Architecturally significant requirements

# [Functional Requirements](https://csuprod-my.sharepoint.com/personal/jfootn01_student_csu_edu_au/_layouts/15/Doc.aspx?sourcedoc=%7B0470BD34-6172-4A43-829F-A43B9494ECB3%7D&file=Functional%20Requirements.docx&action=default&mobileredirect=true)

# 5. Decisions, constraints, and justifications

## Decisions:

* The decision has been made to migrate the project from using Docker Compose to using Kubernetes for container orchestration to allow for easier scaling of the application as the number of users and devices increases.
* The project will use a microservices architecture to enable scalability, flexibility, and maintainability.
* The project will use message queues to enable loose coupling between services and improve fault tolerance.
* The project will use Python as the main programming language as it is an easy to use, powerful and common language, allowing for consistency across the system and ease of maintenance.

## Constraints:

* There is a limited budget for the project, which will affect the choice of tools and services used.
* The team has limited experience with Kubernetes, so there will be a learning curve for the migration.
* The project must be completed within the timeframe of the subject.

## Justifications:

* Kubernetes provides built-in support for fault tolerance and high availability, which will improve the reliability of the application.
* Using a microservices architecture and message queues will allow for more modular and maintainable code, making it easier to add new features and services in the future.

# 6. Architectural Mechanisms

* Microservices – the platform is implemented as a set of microservices, which are to be independently and scalable components that communicate with each other.
* Message Queuing – the microservices communicate with each other using RabbitMQ, a message queue that enables asynchronous communication between services.
* Containerisation – the platform uses Docker to package each microservice and dependencies into a portable unit.
* Load Balancing – the platform uses Kubernetes to employ scalability by increasing or decreasing the number of replicas of a microservice.

# 7. Key abstractions

* Devices – the physical devices that send and receive messages on the IoTa platform.
* Messages – the packets of information that devices exchange with each other on the IoTa platform.
* Microservices – units of functionality that make up the IoTa platform. Each microservice provides a specific set of functionalities.
* Database – the IoTa platform stores device and message data in a database.
* Message Broker – a service that allows microservices and devices to send and receive messages.
* API – the entry point that allows external clients to interact with the IoTa platform.

# 8. Layers or architectural framework

* Microservices - The system is designed as a set of loosely coupled microservices that can be developed, deployed, and scaled independently. Each microservice has a specific responsibility and communicates with other microservices through well-defined APIs.
* Event-driven architecture - The system relies heavily on events and messages to decouple the different components. RabbitMQ is used as a message broker to distribute events between different services.
* Service-oriented architecture - The system is designed as a collection of services that provide specific functionality. Each service is self-contained and can be independently deployed, scaled, and maintained.
* Layered architecture - The system has a layered architecture with a clear separation of concerns. The layers include the presentation layer, application layer, service layer, and data access layer.

# 9. Architectural views

* [**Logical:**](https://csuprod-my.sharepoint.com/personal/jfootn01_student_csu_edu_au/_layouts/15/onedrive.aspx?ga=1&id=%2Fpersonal%2Fjfootn01%5Fstudent%5Fcsu%5Fedu%5Fau%2FDocuments%2F2023%20ITC%20Project%20%2D%20Team%208%2FDocumentation%2FArchitecture%20Diagram%20%2D%20Current%2Epng&parent=%2Fpersonal%2Fjfootn01%5Fstudent%5Fcsu%5Fedu%5Fau%2FDocuments%2F2023%20ITC%20Project%20%2D%20Team%208%2FDocumentation) Describes the structure and behavior of architecturally significant portions of the system. This might include the package structure, critical interfaces, important classes and subsystems, and the relationships between these elements. It also includes physical and logical views of persistent data, if persistence will be built into the system. This is a documented subset of the design.