Non-Functional Requirements Analysis

DPI IoTa Improvements Project

ITC303 - 202330

Group 8 – Clare Strik, Jack Footner & Sam York

# Introduction

The IoTa platform handles is a set of microservices to process the messaging of IoT (Internet of Things) devices managed by the NSW Department of Primary Industries (DPI). DPI has commissioned the project to improve the implementation and deployment of IoTa based on its first 12 months of operation.

This project is essentially several smaller projects rolled into one, which are listed by priority as follows; Migrating from Docker Compose to Kubernetes, Refactoring the MQTT front end processor, adding Prometheus metrics and a Grafana dashboard, improving the device management web app, refactor front end processes to allow them running standalone, fix the RabbitMQ async code, and improving the unit tests.

# System Qualities

## Usability

The system must be easy to use, both for the current developers that are comfortable with the current system and for any new developers in the future to be able to easily understand and start working with the system. The current system requires only installing prerequisites, cloning the repo and running a single command to have a test instance running, the new system should also allow this, i.e., not requiring more than a handful of commands which can be included in the repositories readme.

## Reliability

The current system does have an issue with reliability which will hopefully be addressed by this project. As the system is run using Docker Compose, only a single instance of each container is ever run at a time, by implementing Kubernetes we hope to allow for the orchestration system to provide failover and load balanced container instances. By providing these systems within Kubernetes, if an error causes on container to collapse, its load balanced/failover partner will immediately take over and allow the system as a whole to continue running.

## Performance

The current system sees a limited throughput, with roughly one message received every 1-5 seconds, because of this low rate, performance is not a high priority. However, as the system expands and new devices are brought online, the number of messages is set to increase so the project needs to still be mindful of overall system performance. As stated in the Reliability section above, switching to Kubernetes should allow much easier orchestration of load balancing configurations of the containers, this will allow the system to scale to any higher throughput.

## Supportability

As the system is expanded, the number of devices will increase greatly, each of which often has its own messaging protocol/standard. To allow the implementation and integration of new devices easily into the system the front end messaging processor will be refactored to use a dynamic module system. Using this system allows developers to write simple functions that parse message for given MQTT topics into a native format before being passed to the backend services without needing to rewrite the entire message processing service.

# System Interfaces

## User Interfaces

The service control system only uses a command line as its User Interface. All services currently are started and stopped in a Linux Bash terminal using Docker and Docker Compose commands. The improved system should allow the same low level control that the developers are used to.

There is a device management web app which does feature a GUI any changes to this service must maintain the current design aesthetic.

## Interfaces to External Systems or Devices

The system has two major external system interfaces, the device message inputs and the time series data output.

The device inputs will come in various forms including MQTT and The Things Network, currently these are the only two system inputs but this could change in the future. All input messages must first be transformed into a consistent standard format before being sent further into the process chain.

The main system output is the REST API which is relied upon by the management app and any external agencies accessing data. The new timeseries database will also access the system via this API.

# System Constraints

## Programming Language

The current system is written using Python and some limited JavaScript. The new built system should continue to use Python to keep consistency and as this is the language the current developers are most comfortable using. The limited JavaScript was written out of an ease of library and example code availability for The Things Network processor and should not be used anymore.

# System Compliance

## Licensing Constraints

The system needs to be licensed for government and commercial use and any libraries used must also allow this, as DPI is a state government agency.

## Legal Copyright and Other Notices

All written code must be the own work of Group 8 and the individuals who are a part of it. Any code from other sources must be clearly referenced and licensed according to the previous section.

# System Documentation

Documentation where appropriate should be made with markdown and mermaid to allow the documentation to be easily managed along side the application within the repository itself.