Cloud Architecture Prototype Spec

This project is to create an

# Deliverables

* Android mobile app to search and display sell offers from an offer feed
* A C# .NET secure communications library to upload a set of files to a server
* Design and implementation of a secure server to accept data uploads and manage processing, accept processed results and send them to registered mobile phones
* Graphical test program to use library functionality

Technology

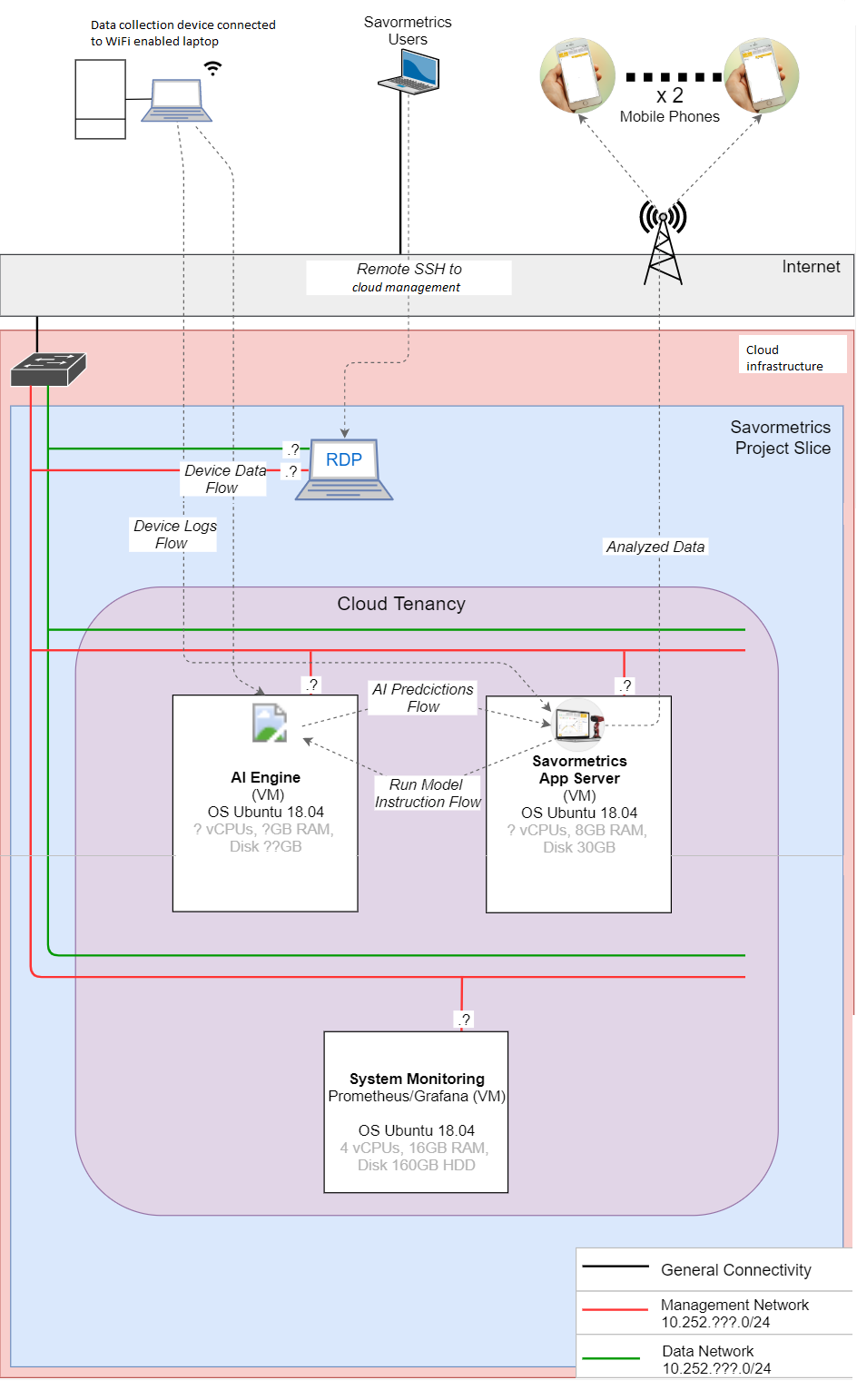
The library will make use of C# and .NET. The mobile app shall be Android, and compatible with the Android version of the current largest install base. The server app may use any technologies which can run on an Ubuntu 18.04 server instance.

Software Development Requirements

The library and its subcomponents should be developed using object-oriented principles and proper design patterns. We will require automated unit tests for all components of the library.

We will also require a test program that makes use of the library functionality. We require a UI-based program using WPF.

# Cloud Architecture and Operation Detail



Secure Communications Library

A data collection GUI created by Savormetrics will be running at a data collection site on a WiFi connected laptop. The device creates a number of files forming one complete set.

The library will contain two functions;

SendFiles(path, timeout, meta1, meta2, …)

The path will be a single file or a folder. All files will be added to an internal queue and the function will return immediately. The function will create a background task to transfer the files in the queue until completed. The internal upload process will confirm, with retries if necessary, that a complete folder has been uploaded. This means, that if a folder path is given, that all files can be considered as a set, and the entire set can be marked as completed.

Meta information will also be sent with the folder of files. There may be a variable number of arguments, either strings, floats, or ints, which is also to be sent with a data set. At this point, we envision only these arguments to be sent, which will be referred to further in this document:

String SellerID

Float Quantity

Float Price

There will be approximately a maximum of 72 files of 7MB each in one data set.

It will return a status code of 0 if the function was successful. An example of not being successful, returning a different status code, is if the file path was not found, or if the machine is not currently online.

If an upload queue is in progress, the function will not accept any new files, and return an error code.

SendCancel()

This function will clear the current queue and send a signal to the server to indicate that the upload was aborted. A completed data set cannot be cancelled.

QueueStatus()

The function will return the status of the queue, such as how many files remain to be uploaded in the set, and how long until the timeout for that set expires.

Communications Test GUI

A test GUI will demonstrate the functionality of the communications library. It will consist of a file browser button, where you can select one or multiple files to be added to be uploaded, and a box to put a timeout value, SellerID, and Quantity. Finally, there will be a button to start the uploading. A status display will show the number of files remaining in the queue are regular intervals.

The status returned from the file upload will also be shown; indicating such errors as the path not found, the file was not accessible, or the internet is not online, or other errors as necessary.

Application Server

The application server is a controller for all the cloud processes. It will:

* Wait indefinitely for a new data upload from the data collection device.
* Accept a complete set of files and confirm their receipt to the communications library
* Allow a cancel message for a particular set, which works in conjunction with the communications library
* Upon receipt of a complete set of files which was not cancelled, make the set of files available as a shared folder to the A.I. engine
* Send a start processing command to the A.I. server, which contains only the path to the data folder.
* Receive a single float value representing a quality rating from the A.I. server and store it together with the meta data for this set
* The files which were processed can now be erased from storage
* A price will be computed based on the quality value, based on a configurable formula. Initially, the formula will be:

Quality Computed Price

>=350 1.50

<350 1.00

* A set of values called an “offer” will be stored permanently in a database and consist of:

String SellerID

String OfferID

Float Quantity

String Quality

Float Price

* The complete offer will immediately be pushed to all registered phones
* The app server will also wait indefinitely for a “buy” request from the mobile app. A buy request consists of:

Integer BuyerID

Integer OfferID

Float Quantity

* The app server will append the buy offer to a csv file, send a Buy response string back to the mobile app (consisting solely of “Accepted”) and take no further action

A.I. Server

The A.I. server waits for data processing requests, then starts the processing with a Python program. It will:

* Accept a start processing request, with a location of a set of data files
* Start a python program with the following shell command:

Python {program name} {data folder}

* Wait for the Python program to complete, which return a status code. If the status code is 0, look for a csv file written to the shared folder. It will be called “result.csv” and consist of one header row and one data row, as follows:

Quality

{float value for quality, e.g. 366.5)

* Send the Quality value back to the App Server

Mobile App

The app will consist of:

* A service which can receive push notifications from the App Server
* Each “offer” will be stored permanently in a local database
* The App GUI will show a list of all current offers
* The user can tap on an offer in the list and tap “Buy”
* The user will enter a Quantity to buy
* The App will send the Buy message to the App Server, as described under Application Server
* The Buy response string from the App Server will be displayed
* The Buy screen will close and return to the Offer pane