IoT Platform Orchestration and Management Tool

MyT Platform

Semester 1 Report

Daemon Macklin

20075689

Supervisor: Joe Daly

BSc (Hons) in The Internet of Things

MyT Platform
IoT Platform Orchestration and Management Tool

Abstract

The Internet of Things(IoT) is a relatively new idea which involves a network of smart devices which share data. IoT is being used in a wide range of applications, such as the Smart home, Smart Agriculture and other fields. All of these applications have the same core ideas. Collect data, Analyse data and Store data, typically this is all done in the cloud, with services such as Wia, Wyliodrin or AWS IoT. MyT aims to find a new way to deploy and manage IoT services in the cloud. By giving the user control of their own IoT platforms at design, creation and deployment so they get the IoT application that best suits them.

Table of Contents

Introduction

MyT - What is it?

MyT is a cloud agnostic orchestration and management tool which allows users to design an IoT platform and then deploy it on a cloud service or server of their choice. Allowing the user to customize some aspects of their platform. For example different database solutions offer different features and are typically designed for different data. If a user has very structured data they will be more suitable for a relational database. On the other hand, if the user does not have structured data, or does not know what data they will be collecting they would be more suited to a NoSQL solution. This will provide an IoT platform that will suit the needs of the user. The MyT application will be a front end to manage the users platforms. Automating parts of maintaining your own IoT stack, allowing the user to focus on the rest of their IoT system.

Why would you use it?

The issue of data ownership is becoming a major concern in modern society. Especially in technology, many companies allow users to utilize their services for “free”. However data they collect is worth more than most users realize. The practice of selling user data is very common, so it is hard to tell who to trust. A solution to that is to trust no one. Other IoT platforms MyT aims to compete with like Wia technically own data you upload to their service and could potentially sell or view that data with little consequence. From the point of view of a person or company who wants to monitor systems and produce sensitive data, owning and being able to confidently secure or even delete that data is vital. By design MyT will put the user in control of their data, so that they have complete ownership of it and can be sure if the data has been deleted it is gone forever.

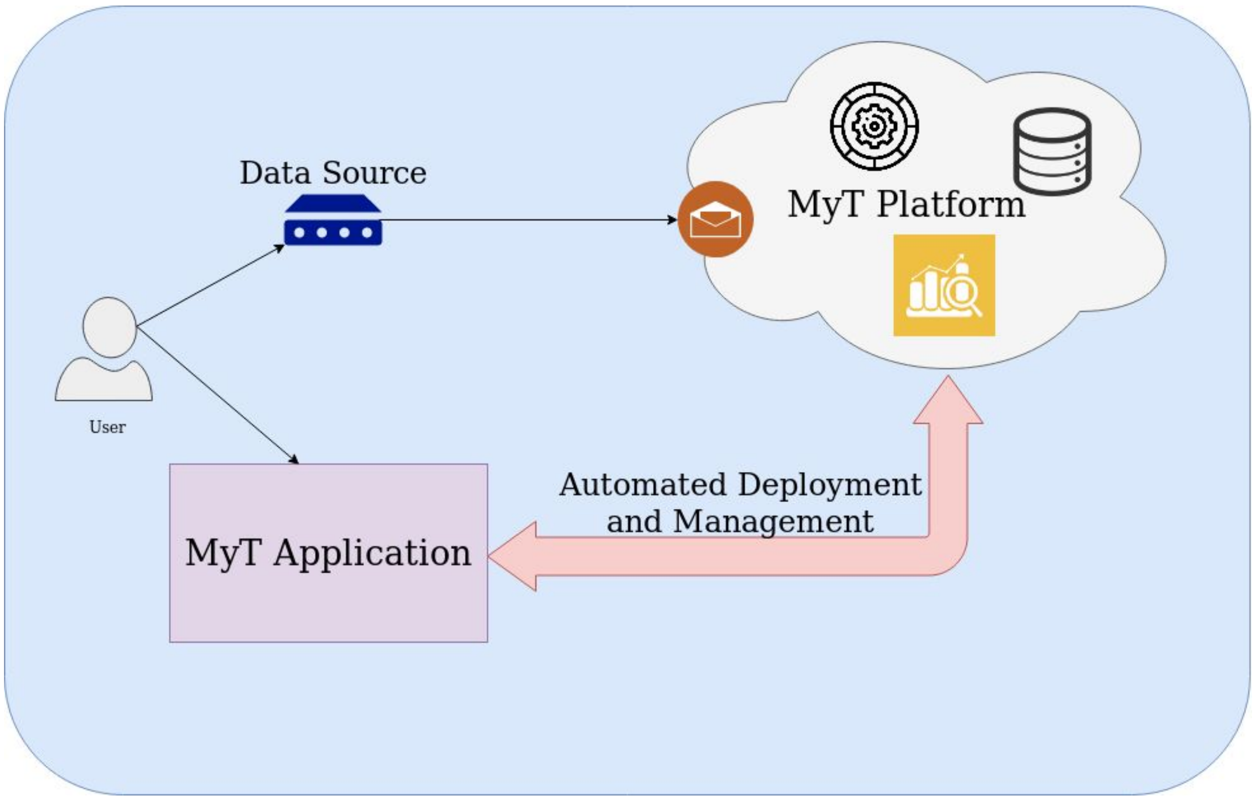
The second motivator is that when you use an IoT platform like Wia for your IoT application you are relying on their service to be working and fully functional at all times. Not only that if the IoT platform you are using goes out of business your IoT application goes with it. MyT aims to make it easy to cut out this third party application. This will give the user control over their IoT platform. Being able to directly manage their platform or fix issues if they occur or blow it up and upload your data to a completely new MyT platform.

How will it work?

MyT has a number of moving parts. The first thing the user will see is the MyT application. This will be the portal where the user can create, delete and manage their IoT platforms. This front end must be always available to the user, so they always have control of their platforms. This application will allow the users to run scripts which will go off and manage their infrastructure. There are a number of tools that could be used for this process such as Cloud-Formation, Terraform, Ansible, etc.

When the user wants to create an IoT platform they will first pick out the technologies they want to use, pick their cloud service provider and hit launch. MyT will then use a combination of orchestration and management tools to create the users desired infrastructure.

Once the user has created their platform. MyT will continue to monitor it, and give the user details about it. When the user wants to take the stack down, MyT will be able to use the same combination of tools it used to create it to terminate the services.



Technologies Required

There are a number of technologies that will be utilized to create the MyT application.

1. Orchestration tool - This will be the tool used to create the Infrastructure.
2. Management tool - This will be the tool used to install applications and setup the infrastructure.
3. Application - An Application will be required to allow the user to run the Orchestration and Management tools.
4. Cloud Service - This will vary from user to user. But for now MyT aims to work with Openstack and Amazon Web Services.

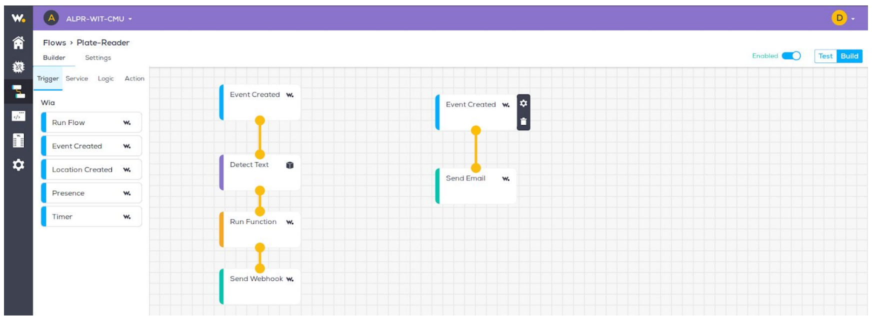
There will likely be more tools required, this is a high level view of what is required. The main requirement that all of the tools must adhere to is that they be open-source.

Existing Technology

Wia

Wia (Wia, 2019a) is an IoT platform that allows the user to easily upload, process, store and graph data. It also supports interaction with a number of different AWS tools such as image recognition. This makes Wia one of the most powerful IoT platforms available. However there are a number of downsides that MyT will aim to improve on.

Using Wia

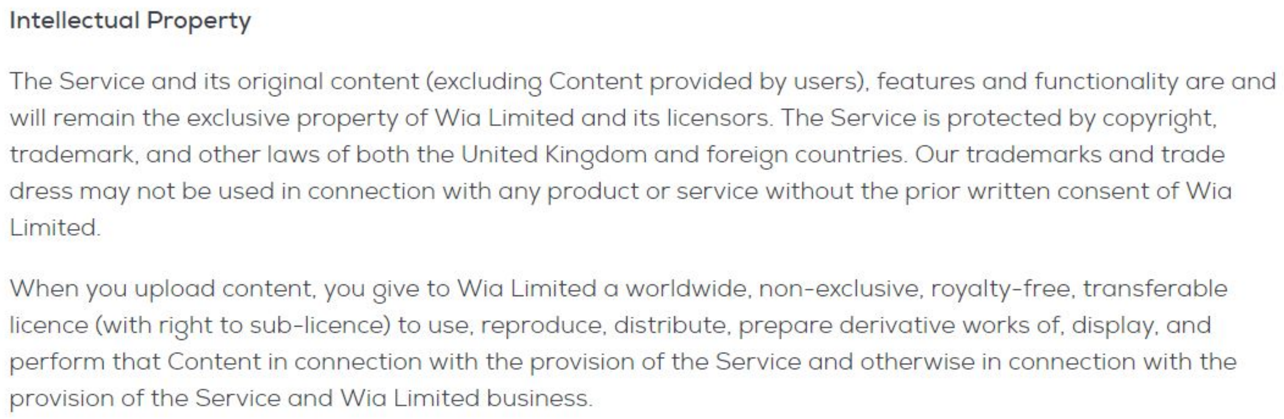


Wia allows you to send data to an endpoint which will upload the data to your “flow”. A “flow” is how Wia allows the user to perform different actions ranging from basic functions, webhooks, sending emails, etc. However there are a few issues.

Firstly from my personal experience the service is not the most reliable. It often had service outages in key components of my applications. All I could do when this occurred was to wait until they were fixed. Sometimes my flows and widgets would be lost for no reason and I would not be able to get them back, so I would lose hours of progress because their service was unreliable.

The second problem was that there were only two ways to execute custom code. The first was to create an external web service and use the webhook component to send data to it, this made my platform unnecessarily complicated. The second way was to use the run function component. This is the easiest way to do it, but it severely limits the type of data processing you can do as you are limited to running basic javascript without external libraries.

The last issue comes back to the data ownership issue, when you use a service like Wia you are trusting them to handle your data. If you were uploading sensitive data there is no guarantee that you have complete control over your data. If you read Wia's terms and services (Wia, 2019b) it is plainly written that any data uploaded to Wia is owned by Wia:

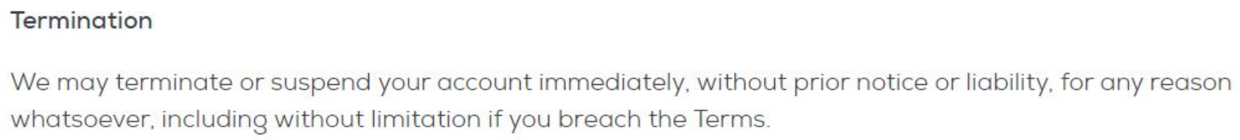


Intellectual Property

The Service and its original content (excluding Content provided by users), features and functionality are and will remain the exclusive property of Wia Limited and its licensors. The Service is protected by copyright, trademark, and other laws of both the United Kingdom and foreign countries. Our trademarks and trade dress may not be used in connection with any product or service without the prior written consent of Wia Limited.

When you upload content, you give to Wia Limited a worldwide, non-exclusive, royalty-free, transferable licence (with right to sub-licence) to use, reproduce, distribute, prepare derivative works of, display, and perform that Content in connection with the provision of the Service and otherwise in connection with the provision of the Service and Wia Limited business.

And that they can terminate your account any time they want:



Termination

We may terminate or suspend your account immediately, without prior notice or liability, for any reason whatsoever, including without limitation if you breach the Terms.

This might be acceptable to a hobbiest, but if you are a business who is collecting sensitive data or if you take data protection very seriously then Wia would not be a suitable solution for an IoT platform.

The MyT Platform

Functional Requirements

A standard IoT platform has four features (Guth et al., 2016):

1. Messaging - IoT platforms typically messaging services like MQTT to listen out for incoming data.
2. Data Processing - Users should be allowed to process the data they collect with custom code before data is stored.
3. Storage - User data will need to be stored securely.
4. Graphing - Users should be allowed to create and view their own graphs to visualize their data.

As well as these the platform will need to perform well for small to medium scale IoT applications, have a completely automated deployment and be secure. All of these issues are affected by the applications used but also impacted by the system architecture.

Messaging

The Messaging broker will be listening for messages from the users devices and sending them to subscribers. The user will not be picking this one out as the security features vary from broker to broker. The user won't be able to select the broker, but can opt in or out of the brokers features. The suitable messaging broker must be open source.

The Messaging brokers I think could be suitable are:

1. Mosquitto (Eclipse Mosquitto, 2019)
2. RabbitMQ (Rabbitmq.com, 2019)

The main difference between these two brokers is the different protocols they use. Mosquitto uses MQTT where as RabbitMQ uses AMQP.

MQTT vs AMQP

The MQTT and AMQP protocols aim to provide the same service, sending messages across a network of devices. However they go about it two different ways. MQTT is more of a lightweight implementation providing a simple way to send data. Where as AMQP has more features built in. MQTT was originally created by IBM in 1999. It was designed in order to transmit data over unreliable, low bandwidth wireless networks to traditional LANs (Hunkeler, Truong and Stanford-Clark, 2008). AMQP comes from the finance industry, who wanted to be able to reliably commit transactions (Subramoni et al., 2008). This has lead to a number of differences between the two. MQTT being more suitable for simple clients talking to servers, but without a lot of built in security or quality of service features. AMQP on the other hand supports a wide range of messaging forms, different levels of acknowledgment, as well as providing a number of different approaches to TLS negotiation. As a result, AMQP is the more feature rich option, which will allow MyT to provide secure, reliable messaging. This is why MyT will be using RabbitMQ as a messaging broker.

Data Processing

Data processing is a key part of IoT, and where services like Wia and Wyliodrin can fall short. As they allow a user to use their infrastructure. Giving a user the freedom to write whatever code they want is a major security concern. As a result, these services might not fit the needs of an IoT application. This is where MyT can improve. By giving the power to the user to write and import whatever code they want. MyT can do this because the user is hosting the platform on their own infrastructure.

Next question is how should we let the user interact with the data. Based on the architecture of the platform. MyT will already need a wrapper to connect to the RabbitMQ broker and the database. All of the data will pass through here so this is an obvious place to allow the user to process the data. This wrapper is going to be written is Python, as it has libraries to handle database connections and listening to RabbitMQ. Python is also a good option as it is one of the most popular languages among developers (Stack Overflow, 2019). So users will be able to inject Python code into this component for data processing.

Storage

The best storage solution is dependant on the type of data that the user is collecting. One of the core features of MyT is the user's ability to customize their platform. The database has the biggest impact on performance of the platform. If a user is collecting structured data and they don't need any flexibility, why give them a database which is flexible but is then less efficient because of it. As a result MyT will let the user pick one of a number of databases which will be best suited to different types of data.

1. MySQL (Mysql.com, 2019) - MySQL is a standard relational database. This database will be used for structured, non time series data.
2. MongoDB (MongoDB, 2019) - MongoDB is the poster child for NoSQL databases. This database will be used for unstructured, non time series data.
3. TimeScale (Timescale.com, 2019) - TimeScale is a plugin for PostGres that will optimize the database for time series data. This database will be used for structured, time series data.
4. InfluxDB (InfluxData, 2019) - InfluxDB is a NoSQL database for time series data. This database will be used for unstructured, time series data.

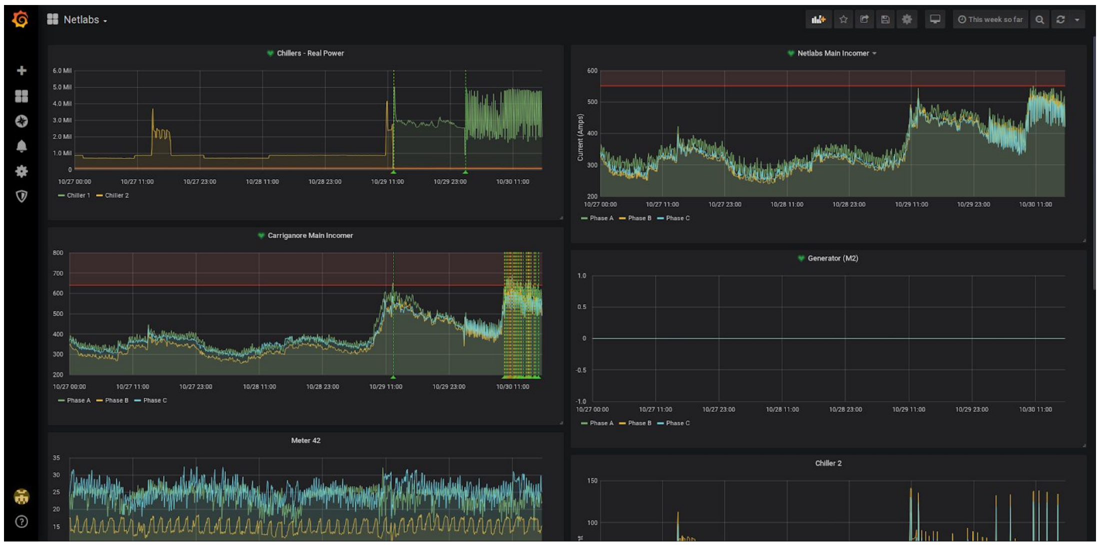
DataBase	Database Model	Initial Release	DB-Engines Ranking (Overall)*	Opensource
MySQL	Relational	1995	2nd	Yes
MongoDB	Document Store	2009	5th	Yes
TimeScale	Time Series/Relational	2017	124th	Yes
InfluxDB	Time Series	2013	33rd	Yes

* (*DB-Engines. 2019)*

As you can see the different database solutions that MyT aims to have by the end of the project will give the user a selection of options that they can use the suit their platform to suit their needs.

Graphing

Graphing is a key part of IoT. Being able to collect and store data is not useful if the user can't visualize their data. MyT will need to allow the users to do this. There are two ways MyT can incorporate graphing. The first is to create a node front end application with a module such as chart.js specifically for the MyT platform. However a lot of this work will go beyond the scope of the project, and making a front end application that will hook into all of the different databases will be time consuming. However, there is an open source application that we can use to do this out of the box: Grafana (Grafana Labs, 2019).



Grafana is an open source data visualization and monitoring tool. It will hook into all of the databases MyT plans to use by default. It will allow the user to create many different graphs and dashboards. It also includes alerting features and has a library of plugins that the user can install. Using Grafana will allow me to focus more of my time on the other features. As well as give the user a better tool for them to visualize their data.

System Architecture

All of the applications discussed above need to be able to work together. There are a number of ways to deploy this system. There are three methods that would be suitable.

1. Centralized
2. Distributed
3. Containerization

Centralized

The centralized method will take all of the applications and put them on one machine. This method makes it easier to connect all of the devices together as there is no need to do much network configuration. As well as that it will also make the platform cheaper to operate as MyT will only be using one machine on your cloud service. However these are the only upsides. With all of the applications being installed onto one system the odds of dependency clashes is very high, you can get around this issue by putting each application in a virtual environment, but this adds extra complexity.

Distributed

This method solves the dependency clash issue with the all in one method. But the downside of this is having to configure network connections between the other virtual machines running on the cloud service. As well as the added network complexity, the platform will also be more costly as each application will have their on instance, or virtual machine running them. The Spread Out method also will prevent MyT from working on a users local server/machine.

Containerization

Containerization is a blend between the previous two methods, using Docker (*Docker. 2019)* it will be possible to have the sandboxed applications in the distributed method with the cost effectiveness of the centralized method. Docker also brings a number of other features to the table. Docker will manage the network between the containers giving us the network simplicity from the centralized method.

Using a tool like Docker-Compose it is easy to design and run a multi container application. Using Docker it also makes it simpler to install all of the applications, as most of them have official Docker images we can pull from DockerHub, giving us the most to date version of the application. I believe this is the best option. It provides an elegant solution.

Data Storage

The data will be stored in the database the user chooses. However regardless of their choice MyT needs to ensure that the data is safe. The first question is where will the data be stored. By default each of the databases will create a folder in the container and save all of the data there. This is problematic as:

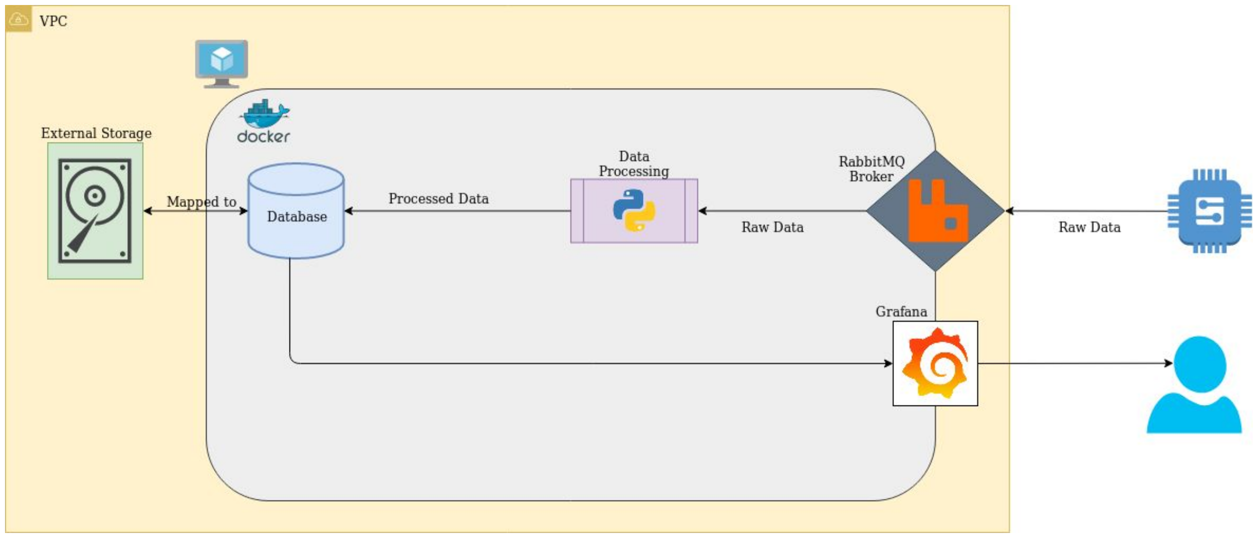
1. The docker container will only have a limited amount of storage.
2. It is bad practice to store data on the root folder of a machine.
3. If the container or the virtual machine fails the data will be lost.

As a result, we need to store data externally. This can be done by using external volumes, block storage, etc. Each of the main cloud service providers gives us the ability to attach external volumes to virtual machines. If data is stored externally the users data will be safe in the event of a virtual machine failing. A recovery service can be developed to get this data back via the application.

Security

Security will be handled in the application, i.e using TLS in RabbitMQ. But we can reduce security risks with a secure architecture design. This is done on a number of levels. The first level is the Network Access Control Layer. On AWS we can only allow certain traffic in and out of our Virtual Network. The next layer is the Security Groups. All of the cloud service providers allow us to only allow certain traffic into a virtual machine. This can be very useful. For example the Secure Shell protocol uses port 22, if we restrict access to this port to only the IP address of the user they will be the only one to be able to SSH onto the virtual machine. This idea can be extended to the other ports.

Architecture Diagram



This is a visualization of the MyT platform. With this architecture which uses Docker to manage the deployment and operation of the application, we have a multi container application that will perform well for small to medium scale IoT applications. Using Docker will also help in making it more achievable to autonomously deploy MyT.

Deployment Strategy

Orchestration vs Management

In cloud computing the terms Orchestration and Management mean two different things. Orchestration refers to the creation of the infrastructure. An Orchestration tool lets developers define what they want their cloud infrastructure to look like in formats such as JSON and YAML, this is known as a domain-specific language. Then tools like Terraform and Cloudformation can interpret that data and create the infrastructure on the cloud service (Liu, Loo and Mao, 2011).

Management on the other hand is how the infrastructure that has been created is configured. The developer defines what applications are installed and how they are configured in formats such as JSON and YAML similar to Orchestration, then tools like Ansible and Salt can interpret that data and install and configure the applications on the target machine or machines (Schroeter et al., 2012). Both principles are commonly referred to as "*Infrastructure as Code*" *(Rahman, Mahdavi-Hezaveh and Williams, 2019)*.

Orchestration

Being able to create cloud infrastructure for a MyT platform to be deployed on will be the job of the Orchestration tool. The one requirement for our Orchestration tool other than it being open source, is that is should be cloud agnostic. Meaning it will work with most cloud service providers.

CloudFormation[1]/Google Cloud Deployment Manager[2]/Microsoft Azure Automation[3]

Each of the big 3 cloud service providers have their own Orchestration tool. Each of which provide a full suite of tools for automating the creation of cloud infrastructure. However there a number of drawbacks to using these tools.

1. These tools are not cloud agnostic.
2. These tools are not open source.
3. To have a cloud agnostic application I will need to develop the same Infrastructure as Code using three different tools/languages.

[1] (Amazon, 2019) [2] (Google Cloud, 2019) [3] (Azure.microsoft.com, 2019)

Terraform vs Pulumi

Terraform, developed by HashiCorp (Terraform by HashiCorp, 2019) is the industry standard for cloud agnostic orchestration tools. Being able to be used on just about any cloud service provider. Expressed in JSON to define the infrastructure that we want to deploy then terraform will take care of the rest.

Pulumi (pulumi, 2019) is a new tool which aims to be a competitor to Terraform. It claims to do everything Terraform can do, but instead of using JSON you write your IAC in Python/Javascript or a number of other languages similar to using Boto for AWS.

From experimenting with both, I found that Pulumi only has one advantage over Terraform. Since you are writing code to create the infrastructure you have all the tools you would normally have with that language, particularly conditionals. Terraform currently has no “if” conditional. However this issue can be avoided by using a tool like Jinja to dynamically create infrastructure based on user input.

Terraform on the other hand has a number of benefits over pulumi. Firstly, it is just easier to use. With Pulumi you have to sign in to an account to use it. Then despite the fact that you are writing normal code for your infrastructure you still need to use the Pulumi command line interface to run your scripts. There is also less to get started with Terraform, just install the executable, write the Terraform files and execute the “apply” command.

Pulumi is a lot more work for the same outcome compared to Terraform. Terraform also has a lot more documentation as it has been around longer, Terraform supports more providers, and it will be easier to run terraform scripts autonomously. As a result I will be using Terraform as my Orchestration tool.

Management

Once Terraform has created the networking, virtual machines and storage in the cloud the next step will be to install the MyT platform onto the infrastructure. This task will be done by the management tool. The management tool that will be used must be lightweight and makes it easy to install and configure remote servers.

Chef[1]/Puppet[2]/Salt[3]

Chef, Puppet and Salt are industry standard for configuration management tool. They allow for users to install and configure packages on remote servers by linking them to a master server. However the issue they have is that there is a lot of setup necessary before users can get to a point where you are simply installing packages. These tools allow for a lot more however, MyT will only be using the management tool to set up the server and install Docker, so there isn't a need for much more.

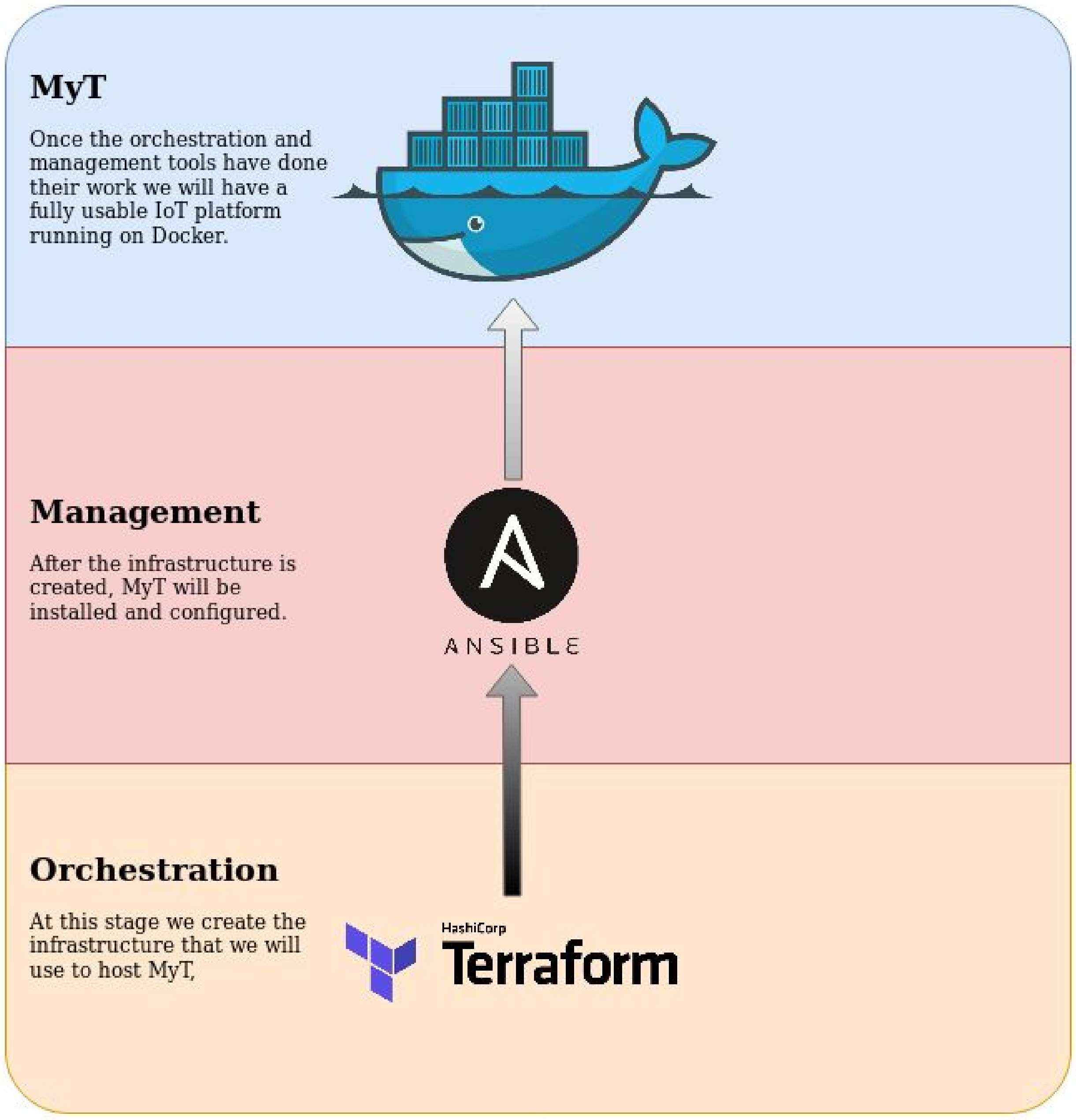
[1] *(Chef, 2019)* [2] (Puppet.com, 2019) [3] (SaltStack, 2019)

Ansible

Red Hat Ansible (Ansible, 2019) is another industry standard for configuration management. Except Ansible does it differently. Instead of linking remote servers to a master server, Ansible allows the user to point at the IP address of any machine or machines and then Ansible will install the software defined in the playbook. Making it a "radically simple IT automation engine".

Ansible uses playbooks, written in YAML to allow the user to define what they want their machine to "look like". The playbook consists of a config file which tells Ansible which ssh key to use and some other settings. An inventory file, this contains the IP address of list of IP addresses of the machines to configure, and the main file. This main file typically will call a number of roles for Ansible to execute. The Ansible role is where the users defines what changes should be made to the machine, which packages should be installed, what files should be where, etc. Ansible does this by taking the playbook and converting them to resource models of the desired state of the machine. Ansible will then execute these models on the remote machine over ssh. The only thing required on the remote machine is Python. The power and simplicity of Ansible makes it the most suitable tool for MyT.

Deployment Strategy Summary


MyT
Once the orchestration and management tools have done their work we will have a fully usable IoT platform running on Docker.
Management
After the infrastructure is created, MyT will be installed and configured.
ANSIBLE
Orchestration
At this stage we create the infrastructure that we will use to host MyT,
HashiCorp
Terraform

This is what the deployment strategy will look like. Using Terraform and Ansible we can deploy MyT to any cloud service the user chooses with the press of a button.

MyT Application

Functional Requirements

The MyT application will be the tool the users will use to customize and deploy their IoT platform. The main requirements for the application are:

1. Create users
2. Allow users to save cloud service credentials
3. Allow users to create platforms
4. Allow users to monitor platforms
5. Allow users to download data from their platforms
6. Allow users to delete platforms

The application should also be always be available to the user, so that they can always interact with their platforms. Since the user is storing their highly sensitive credentials MyT will need securely store user data. On top of all of this application must be simple to install.

Application Technologies

To ensure that the user does not have to rely on a third party service, other than the cloud provider they choose for their IoT platform. The application must be available at all times. This would suggest that a desktop application would be a more applicable option. There are a number of desktop application technologies MyT can use but one of the most interesting one is Electron. However there are a number of reasons why a web application would be more suitable.

Electron (Desktop Application)

Electron (Electronjs.org, 2019) essentially boils down to a standard web application that is run on it's own chromium instance. The developers claim that this makes an Electron application system agnostic. The other upside to using Electron is that it is just a wrapper for a normal node application, meaning the MyT application can use whichever framework is most suitable. This does limit the application to being written in node.

Web Application

A web application at first sounds like it will not meet the requirement of having it always available to the user. However this depends on how the application is deployed. Having a simple web application that the user installs on their own machine and is accessed through localhost opens the application to be used in more than one setting. For example services such as moodle allow the user to install their own moodle instance that can be accessed through localhost, but by installing it on a server and using a program like nginx it can all of a sudden be accessed by a number of users who can use the service.

Creating the MyT application as a web app that the user installs themselves allows the application to be available to the user whenever they need it. It also allows it to be used as a service for multiple people to use at one. As a result the MyT application will be a web application that the user installs themselves as this increases the potential applications of MyT.

Application Components

Using a web application does increase the complexity of the MyT application as a whole. There are now three separate parts to the application. The frontend, backend and database components. Each of them have their own requirements but the main one is that there should be an easy way for a user to deploy them.

Frontend

There are a number of web frameworks that MyT can utilize but the two most popular at the moment is Vue (Vuejs.org, 2019) and React (Reactjs.org, 2019). Both of them provide a large tool set for developing front end applications. Both utilize a virtual DOM and provide reactive and composable views. They can both provide a multi-page application which is aesthetically pleasing. As frontend development isn’t a main feature of this project I think it would be best if MyT used Vue, as I have previous experience with it and it will allow me to focus more on more important features.

Backend

The MyT application backend will serve two purposes. Managing user data and interacting with Terraform and Ansible. There are a number of technologies which are available that can do both such as Node.js (Node, 2019) and Python Flask (Pallets, 2019). These two technologies are widely used in modern web application and API development.

Node

Node is an asynchronous event-driven Javascript runtime environment. Allowing developers to write Javascript code outside of a web browser, and use it for server-side scripting. Using Node would make MyT adhere to the “Javascript Everywhere” paradigm. Having both frontend and backend in Javascript. However the issue with Node is the lack of supported libraries available to integrate with Terraform or Ansible.

Python Flask

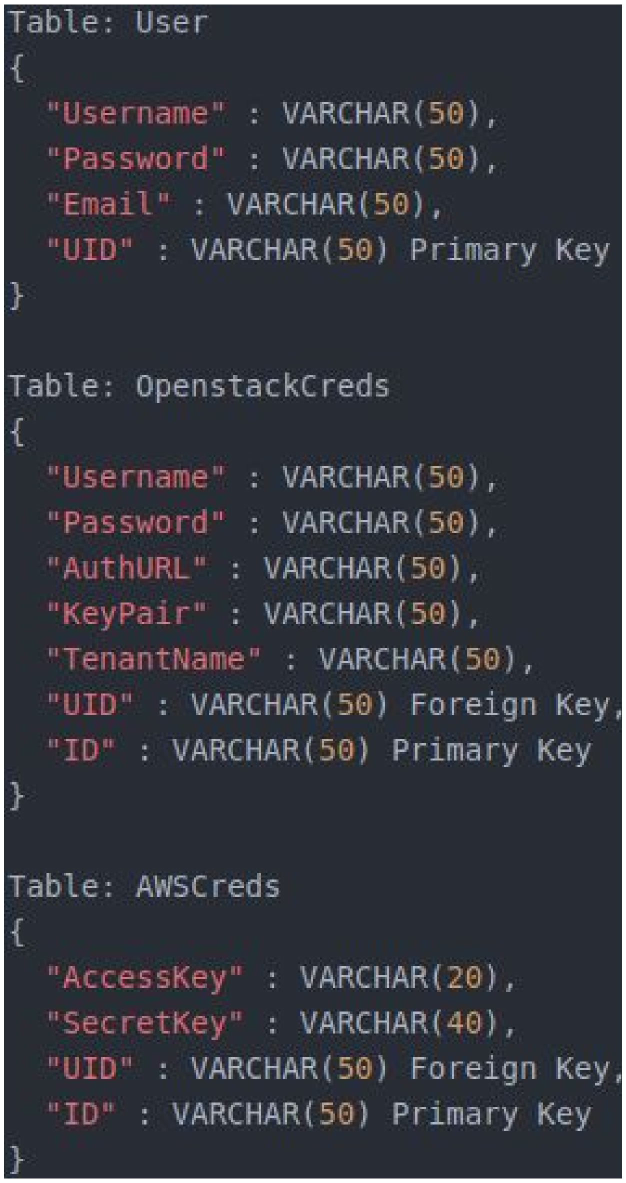
Python Flask provides a simple way to create an API using Python. This complicates the MyT application as there would be two different technologies used for the frontend and backend. However the advantages given by Python outweigh these complications. Python provides a number of libraries for interacting with Terraform and Ansible. Due to this and in my opinion Python is easier to work with, the MyT Application Backend will use Python Flask.

Storage

In order to store user data safely and securely the MyT application will require a database. There are a number of options that MyT could use, Mongo, MySQL, the list goes on. However the aim of this application is to be simple. As a result one database in particular stands out above the rest: SQLite (Sqlite.org, 2019). The flat file database suits the MyT application as the application firstly does not require a lot of database power. The queries executed will only be getting user data from a small database, containing a small number of users and their credentials. Secondly using SQLite is easy to use. The only thing required to use it is the installation of a Python Package.

Sample Data

The SQLite database will have to store a number of things. Firstly the user data, username, password etc. But the second thing will be the users credentials for the various cloud services. SQLite is a relational database, so the number of tables will depend on the number of supported cloud services.



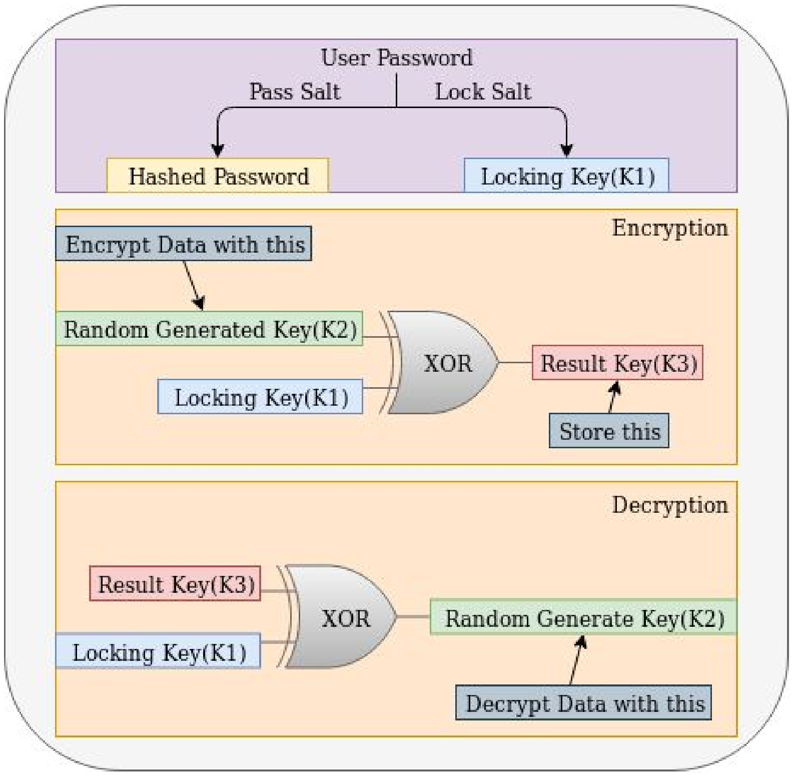
```
Table: User
{
  "Username" : VARCHAR(50),
  "Password" : VARCHAR(50),
  "Email" : VARCHAR(50),
  "UID" : VARCHAR(50) Primary Key
}

Table: OpenstackCreds
{
  "Username" : VARCHAR(50),
  "Password" : VARCHAR(50),
  "AuthURL" : VARCHAR(50),
  "KeyPair" : VARCHAR(50),
  "TenantName" : VARCHAR(50),
  "UID" : VARCHAR(50) Foreign Key,
  "ID" : VARCHAR(50) Primary Key
}

Table: AWSCreds
{
  "AccessKey" : VARCHAR(20),
  "SecretKey" : VARCHAR(40),
  "UID" : VARCHAR(50) Foreign Key,
  "ID" : VARCHAR(50) Primary Key
}
```

Here is an example of what the tables might look like. The user will have their data stored in one table, and then they can have any number of credentials with each of the supported service providers. The credentials tables will store the credentials along with the UID of the user they belong to and only the user with the corresponding UID will be able to view them.

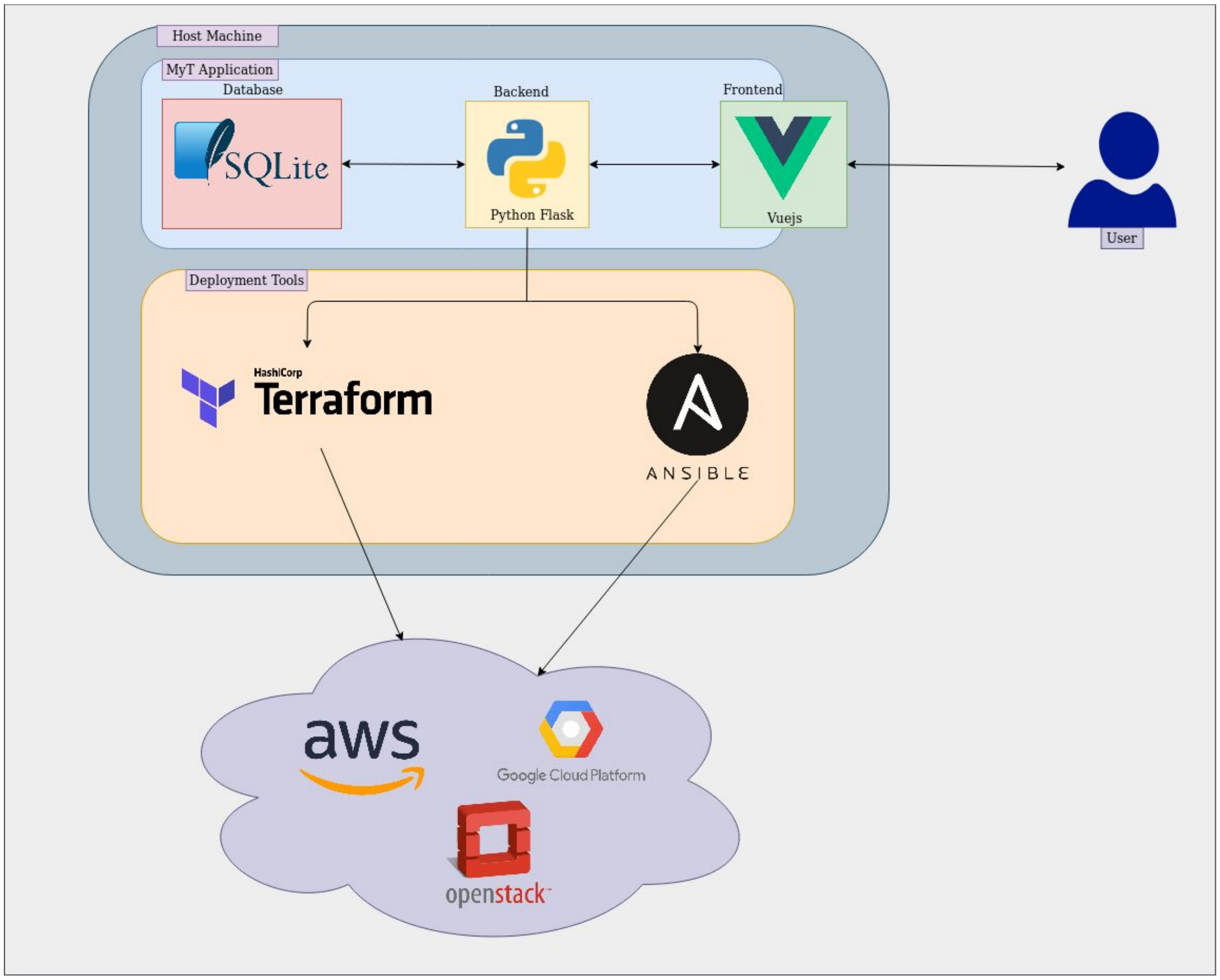
Security

MyT application will still require some sort of encryption method in order to keep the users credentials secure. The user password can easily be hashed using standard password storage practices (Arias, 2019). However the users credentials like the AWS secret key will need to be encrypted to keep the data secure but also be able to be decrypted so that the application can access their credentials. There are also a set of best practices for doing this too.

Encrypting user data can be done with three keys. The first step is to generate a “Locking Key” from the user password, and storing the salt used to generate it. Then a random key is generated, this random key will be used to encrypt the user’s credentials. To store this key securely we XOR the locking key with the random key. This result can now be stored. When the application needs to decrypt the credentials we can recreate the locking key by asking for the user’s password then XOR the generated locking key with the stored key, this result will be the Random key that was used to encrypt the data.



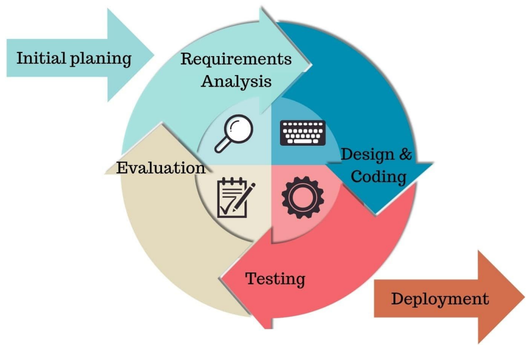
Application Deployment



This MyT application will be how the user will create their platform. Allowing the user to pick which database they want to use along with some security options. The application will be installed on one machine, via a Debian package making it easy to install on Linux machines. Depending on how development goes it will either all be in one package or split up into two packages, one for the frontend binaries and one for the backend binaries. Obviously the user will need to install Terraform and Ansible, but both of these can be installed with one command in the Linux terminal if the application cannot do it. Alternatively the user could build the project from source. Or install via a docker image created for the MyT application.

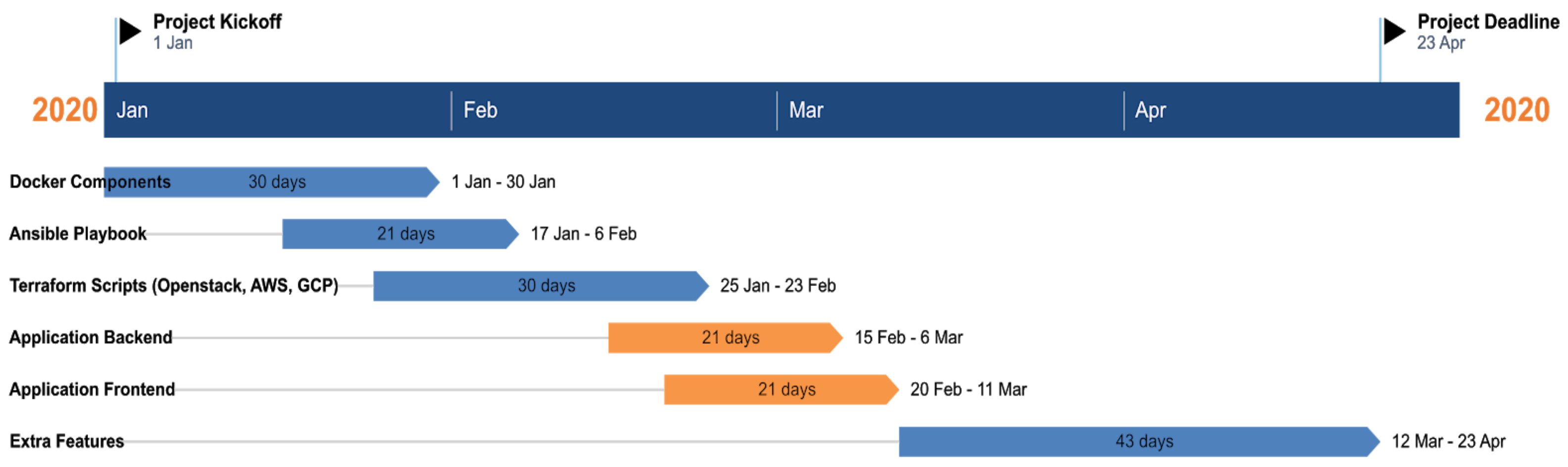
Development

Development Methodology



The MyT platform project has a lot of different components which will rely on each other. As a result the iterative development methodology (Beck, K., et al, 2001) would be suitable for a project of this nature. Both the application and the Infrastructure as code can be developed simultaneously once a baseline for both has been created. Then the process of requirements gathering, implementation, testing and then final evaluation can be done for individual features until the project is ready for release.

Development Plan



Use Cases

“Hobbyist”

Randy Johnson is a web designer. He is very interested in monitoring the temperature in the different rooms in his house. Randy investigates a number of Iot platforms such as Wia and Wyliodrin. Randy likes the simplicity of the platforms but does not like the basic graphing functionality that each service provides. Randy wants to be able to create complex graphs which overlay different sensor readings.

After further research Randy comes across MyT on GitHub. He reads the details of the application and likes the idea of being in complete control of his data. Being able to write complex data processing scripts natively. Most importantly for him he is drawn to the graphs he can create with Grafana. He signs up for an AWS account and installs the application on his laptop.

Randy sets up his AWS credentials on his account and goes through the installation process. He chooses to use a MongoDB as he is not too sure about the data he is going to be collecting. He presses install and his platform is created on his AWS account. The application gives him the MQTT endpoint to send his data to.

Randy then starts to setup his devices to take ambient temperature data and send it to the MQTT endpoint given to him from the MyT application. He then discovers a slight problem. The temperature data is just the raw resistance value instead of the actual temperature value. Instead of rewriting the code on each of his devices to fix this. He decides to write some Python code to convert the data on the MyT platform. The MyT application gives Randy the template code to use. He adds his data conversion code to the template and uploads is via his MyT application to his platform. Now raw data sent to the platform is being converted to actual temperature data and that is stored. Randy then spends the rest of his evening creating the graphs he wanted on his grafana instance.

“Professional”

Kathleen Antonelli is a systems administrator at a research institute. She is in charge of creating and managing infrastructure for various research projects. Most of these projects have the same requirements. Data storage, cloud processing and a way to monitor and analyse data. Like most systems administrators, Kathleen likes to automate what is automatable, and make automatable what is not. As a result, Kathleen wants an opensource tool which will allow her to easily spin up systems to handle the researchers requirements.

After some research Kathleen finds MyT’s github page. Kathleen is interested in it and decides to install it onto one of the companies servers to try it out. She likes what MyT can do for her and the researchers. MyT provides a service to allow Kathleen’s users to create research platforms on the companies Openstack cluster, or an external cloud service whenever they need them. It also gives the users a place to easily manage their platforms, from updating the cloud processing code, to downloading backups of the database for research.

Kathleen can now relax with the knowledge that another part of her job has been automated away. As most of the small to medium sized research projects now use their own instances of the MyT platform.

“Enterprise”

Tom Jones is an automation engineer at a production factory. He is in charge of a large automated production line. Tom wants to be able to monitor and store a number of metrics on the different machines on the production line. Tom’s first concern is that the data on his production line is very sensitive and his bosses want to be sure that the data collected is entirely owned by the company.

After some research Tom finds the MyT website. He likes what MyT can provide because It allows Tom to create a platform to handle all of the production lines data without having to create one from scratch. As well as providing a full cloud based IoT service with no ambiguity about who owns the uploaded sensor data.

Tom has no experience with using cloud services. But the company he works for has a contract with Microsoft which provides the Microsoft Office tools, but also provides access to Microsoft Azure. Tom gets the credentials to the companies Azure account and creates a premium account on the MyT website. After paying a small fee Tom is able to use the MyT premium service which will create his IoT platform on his Azure account. The premium account also gives the company a support line if they ever have issues with their platform.

With their platform deployed on their Azure account, all of the data that is collected and stored is completely owned by Tom’s company.

Appendix

Supervisor Meeting Minutes

Week 1:

Date: 8/10/19
Time: 12:15
Attendance: Daemon Macklin, Joe Daly
Report on previous week's work:
N/A

Plan for upcoming weeks work:

- Do "unstructured" research on Deployment tools, Decision Trees and Desktop applications.
- Look at functional and nonfunctional requirements.

Additional Notes:

Week 3:

Date: 15/10/19
Time: 12:15
Attendance: Daemon Macklin, Joe Daly
Report on previous week's work:

- Showed Electron-Vue proof of concept application.

Plan for upcoming weeks work:

- Project report - Justification for Tech used, Cloud architecture, Deployment strategy, front end.

Additional Notes:

Week 4:

Date: 22/10/19

Time: 12:15

Attendance: Daemon Macklin, Joe Daly

Report on previous week's work:

- Project report headings.

Plan for upcoming weeks work:

- Literature Review.
- Project report - Abstract and introduction.
- Project report - Cloud deployment technologies.
- Project report - Cloud architecture technologies.
- Project report - Front end Desktop vs Web App.

Additional Notes

Week 5:

Mid Term Break

Week 6:

Date: 5/11/19

Time: 12:15

Attendance: Daemon Macklin, Joe Daly

Report on previous week's work:

- Discussed progress on report.

Plan for upcoming weeks work:

- Make Writing style of report consistent - 3rd person.
- Compare Terraform and Pulumi.
- Add references.

Week 7:

Date: 12/11/19
Time: 12:15
Attendance: Daemon Macklin, Joe Daly
Report on previous week's work:

- Discussed progress on report.

Plan for upcoming weeks work:

- Plan for what the proof of concept demo will be
- Who would use MyT and why?
- Speak to potential users
- Development methodology
- Gantt chart

Additional Notes

Week 8:

Date: 19/11/19
Time: 12:15
Attendance: Daemon Macklin, Joe Daly
Report on previous week's work:

- Chosen technologies for MyT

Plan for upcoming weeks work:

- User data security.
- Potential Applications of MyT

Additional Notes

Week 9:

Date: 26/11/19

Time: 12:15

Attendance: Daemon Macklin, Joe Daly

Report on previous week's work:

- Met with Jason Berry to discuss project

Plan for upcoming weeks work:

- Use cases.
- Being reviewing paper.
- State diagram of entire project.
- Start working on proof of concept.

Additional Notes

Week 10:

Date: 5/12/19

Time: 12:15

Attendance: Daemon Macklin, Joe Daly

Report on previous week's work:

- Report first draft finished.

Plan for upcoming weeks work:

- Revise report.
- Format references correctly.

Additional Notes

References

Amazon (2019). AWS CloudFormation - Infrastructure as Code & AWS Resource Provisioning. [online] Amazon Web Services, Inc. Available at: https://aws.amazon.com/cloudformation/ [Accessed 29 Dec. 2019].

Ansible, R. (2019). Ansible is Simple IT Automation. [online] Ansible.com. Available at: https://www.ansible.com/ [Accessed 29 Dec. 2019].

Arias, D. (2019). Hashing Passwords: One-Way Road to Security. [online] Auth0 - Blog. Available at: https://auth0.com/blog/hashing-passwords-one-way-road-to-security/ [Accessed 29 Dec. 2019].

Azure.microsoft.com. (2019). Azure Automation – Cloud Automation Service | Microsoft Azure. [online] Available at: https://azure.microsoft.com/en-us/services/automation/ [Accessed 29 Dec. 2019].

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R. and Kern, J., 2001. Manifesto for agile software development.

Chef. (2019). Chef Infra | Chef. [online] Available at: https://www.chef.io/products/chef-infra/ [Accessed 29 Dec. 2019].

DB-Engines. 2019. Ranking. [ONLINE] Available at: https://db-engines.com/en/ranking. [Accessed 30 October 2019].

https://grafana.com/ [Accessed 29 Dec. 2019].

Docker. 2019. Home. [ONLINE] Available at: https://hub.docker.com/. [Accessed 30 October 2019].

Eclipse Mosquitto. (2019). *Eclipse Mosquitto*. [online] Available at: https://mosquitto.org/ [Accessed 29 Dec. 2019].

Electronjs.org. (2019). Electron | Build cross platform desktop apps with JavaScript, HTML, and CSS.. [online] Available at: https://electronjs.org/ [Accessed 29 Dec. 2019].

Google Cloud. (2019). Cloud Deployment Manager | Google Cloud. [online] Available at: https://cloud.google.com/deployment-manager/ [Accessed 29 Dec. 2019].

Grafana Labs. (2019). *Grafana: The open observability platform*. [online] Available at: https://grafana.com/ [Accessed 29 Dec. 2019].

Guth, J., Breitenbucher, U., Falkenthal, M., Leymann, F. and Reinfurt, L. (2016). Comparison of IoT platform architectures: A field study based on a reference architecture. *2016 Cloudification of the Internet of Things (CIoT)*.

Hunkeler, U., Truong, H. and Stanford-Clark, A. (2008). MQTT-S &#x2014; A publish/subscribe protocol for Wireless Sensor Networks. *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*.

InfluxData. (2019). *InfluxDB: Purpose-Built Open Source Time Series Database | InfluxData*. [online] Available at: https://www.influxdata.com/ [Accessed 29 Dec. 2019].

Liu, C., Loo, B. and Mao, Y. (2011). Declarative automated cloud resource orchestration. *Proceedings of the 2nd ACM Symposium on Cloud Computing - SOCC '11*.

MongoDB. (2019). *The most popular database for modern apps*. [online] Available at: https://www.mongodb.com/ [Accessed 29 Dec. 2019].

Mysql.com. (2019). *MySQL*. [online] Available at: https://www.mysql.com/ [Accessed 29 Dec. 2019].

Node. (2019). *Node.js*. [online] Node.js. Available at: https://nodejs.org/en/ [Accessed 29 Dec. 2019].

Pallets. (2019). *Flask*. [online] Available at: https://palletsprojects.com/p/flask/ [Accessed 29 Dec. 2019].

pulumi. (2019). Pulumi - Modern Infrastructure as Code. [online] Available at: https://www.pulumi.com/ [Accessed 29 Dec. 2019].

Puppet.com. (2019). [online] Available at: https://puppet.com/ [Accessed 29 Dec. 2019].

Rabbitmq.com. (2019). *Messaging that just works — RabbitMQ*. [online] Available at: https://www.rabbitmq.com/ [Accessed 29 Dec. 2019].

Rahman, A., Mahdavi-Hezaveh, R. and Williams, L. (2019). A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108.

Reactjs.org. (2019). *React – A JavaScript library for building user interfaces*. [online] Available at: https://reactjs.org/ [Accessed 29 Dec. 2019].

SaltStack. (2019). Home - SaltStack. [online] Available at: https://www.saltstack.com/ [Accessed 29 Dec. 2019].

Schroeter, J., Mucha, P., Muth, M., Jugel, K. and Lochau, M. (2012). Dynamic configuration management of cloud-based applications. Proceedings of the 16th International Software Product Line Conference on - SPLC '12 -volume 1.

Sqlite.org. (2019). *SQLite Home Page*. [online] Available at: https://www.sqlite.org/index.html [Accessed 29 Dec. 2019].

Stack Overflow. (2019). Stack Overflow Developer Survey 2019. [online] Available at: https://insights.stackoverflow.com/survey/2019#most-loved-dreaded-and-wanted [Accessed 30 Dec. 2019].

Subramoni, H., Marsh, G., Narravula, S., Ping Lai and Panda, D. (2008). Design and evaluation of benchmarks for financial applications using Advanced Message Queuing Protocol (AMQP) over InfiniBand. 2008 Workshop on High Performance Computational Finance.

Terraform by HashiCorp. (2019). Terraform by HashiCorp. [online] Available at: https://www.terraform.io/ [Accessed 29 Dec. 2019].

Timescale.com. (2019). [online] Available at: https://www.timescale.com/ [Accessed 29 Dec. 2019].

Vuejs.org. (2019). *Vue.js*. [online] Available at: https://vuejs.org/ [Accessed 29 Dec. 2019].

Wia. (2019a). *Wia | Helping companies build and deploy Internet of Things solutions*. [online] Available at: https://www.wia.io/ [Accessed 29 Dec. 2019].

Wia. (2019b). *Terms & Conditions* | *Wia*. [online] Available at: https://www.wia.io/terms [Accessed 29 Dec. 2019].