

**MADE
SMARTER**

Servitisation – an overview and how-to guide for industry

Part of the Made Smarter
servitisation demonstrator
June 2022

In collaboration with



CATAPULT
Digital

Contents

3 Section 1: Servitisation overview

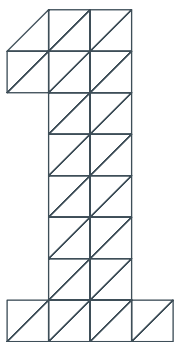
- 4 What is servitisation?
- 4 What is a servitisation demonstrator?
- 4 The value of IoT technologies to servitisation
- 5 The relevance of ecosystems
- 5 The Made Smarter servitisation demonstrator
- 7 The servitisation platform technology stack

8 Section 2: Component selection

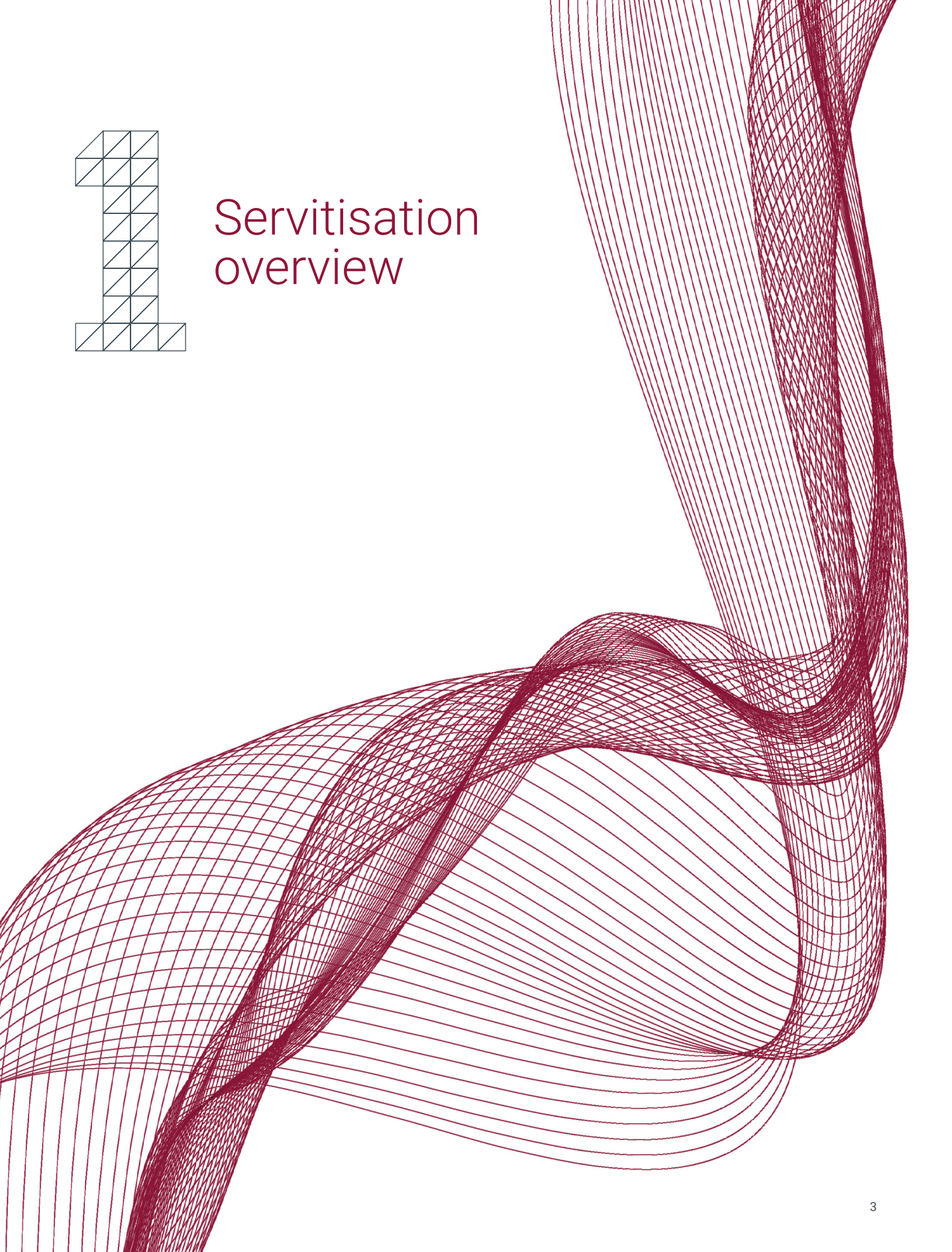
- 9 Component selection roadmap and criteria

13 Section 3: How-to guide

- 14 Servitisation demonstrator components
- 14 Demonstrator component details
- 19 Servitisation demonstrator architecture and step-by-step guide
- 28 Mobile application development
- 29 Business application configuration



Servitisation overview



What is servitisation?

The current economic environment, characterised by globalisation and the pressure of competition and uncertainty, has led many manufacturing businesses to seek customised and personalised solutions/services for customers. This has led to an increased desire to achieve a higher level of customer satisfaction by incorporating and providing integrated solutions and services: servitisation.

Servitisation is the process of increasing value by adding services to products to provide customers a complete product-service system.

There are three types of servitisation:

- product-orientated
- use-orientated
- results-orientated

Ensuring that a customer is provided with the most appropriate servitisation model to achieve their business objectives is of key importance to successful and effective adoption. This means that collection of the right type of data is of significant value for new servitised propositions.

What is a servitisation demonstrator?

A servitisation demonstrator is a rapid prototyping environment that enables companies that are exploring servitisation to identify the key digital components that will bring their new business model to life. The solution offers agile sprints to onboard assets, and provides the functionality required for a proof of value (PoV) /minimum viable product (MVP).

The demonstrator demystifies the huge range of technology options available (protocols, applications, IoT platform, hardware and so on) by showing a best practice approach to developing connected products and services. These are then assembled to build a servitisation platform.

The value of IoT technologies to servitisation

Manufacturers are increasingly seeking to compete with their market peers by offering new services in addition to their products. Creating and capturing value is an essential aspect in servitisation as it drives buying decisions. Manufacturers' use of internet of things (IoT) technologies can be leveraged to assess the potential value of servitisation, as well as significantly enhance its impact.

IoT devices can add value to any servitisation platform and the wider ecosystem:

CREATING NEW VALUE-ADDED SERVICES

IoT enables integration within enterprises for onboarding new services. Companies adopting IoT for servitisation can benefit greatly from services developed by other companies, as they can be included to create added value, or enhance an existing service to improve customer experience, operation and satisfaction.

REDUCING COST AND OPTIMISING RESOURCES

Costs can be cut by sharing a major slice of the infrastructure between multiple assets, using common resources. Furthermore, assets can be tracked using GPS throughout the servitisation journey and monitored using real-time data. This is beneficial for supply chain management of the asset and its parts from production to destination, and provides customer and business transparency in asset tracking.

INCREASING PRODUCTIVITY AND REDUCING TIME

IoT can add value to a given servitisation model by increasing productivity by transmission of asset data using sensors, enabling services to address issues or maintenance in advance or even in real-time. This also reduces the time needed to identify and fix an issue. The IoT technology stack can also help in managing core and additional services, which increases productivity and efficiency.

CUSTOMER LOYALTY AND BETTER USER EXPERIENCE

Customer loyalty can be greatly enhanced by collecting historical data to support marketing automation to reach a much larger pool of customers with better and more advanced services. Improving security, remote monitoring, device management of the assets and data governance using the IoT technology stack also increases customer experience and trust, thus improving scores for customer satisfaction, retention and churn; average resolution time; and customer referral rates.

The relevance of ecosystems

Servitisation requires businesses to work with broader value chains with strong collaboration between providers – the strength of the value being provided is directly related to the strength of the ecosystem. Ecosystems offer competitive advantage to businesses, particularly within servitisation models, as they open new vertical and horizontal markets. Additionally, ecosystems accelerate speed to market, as digitalisation and digital business models require a broad set of skillsets for navigation and orchestration of multiple industry players. Ecosystem partnerships reduce this need, as they enable businesses to fill gaps in capability and expertise.

For end users, servitisation establishes new relationships with the business and also the ecosystem value chain. There is more emphasis on the customer experience and customer journey which inevitably means more service oriented agreements and puts the emphasis on businesses to take more responsibility for their products and services.

There are other key factors that should be considered when building an IoT servitisation platform for the wider ecosystem:

- market readiness for the platform, market volatility, cost and revenue streams
- global rules and regulations regarding products and services
- accurate categorisation of the services being offered and of ecosystem partners for better management of services. This includes aspects such as the identification of differences between basic services, intermediate services and advanced services

- strategy for the digital service being offered and scalability of the technology
- security strategy within the ecosystem and along the value chain
- shared digital tools for transparency, information sharing and benchmarking
- interoperability of the technology with existing operational systems
- the need to develop a value network and evaluate risk and management through service level agreements (SLAs)
- differentiation between additional services, a unique service proposition, and offering the opportunity to a customise services
- maturity of digital and product teams

The Made Smarter servitisation demonstrator

The demonstrator detailed in Section 3 of this report is modelled around a heating-as-a-service servitisation project, using pumps and compressors. This project aimed to help adopters understand the technology required to leverage a servitisation business model.

The key purpose of the project was to build a servitisation demonstrator to showcase:

- the advanced technology stack required to build a servitisation demonstrator and the value of various components in that technology stack
- how various components in the technology stack can be put together to make the servitisation platform operate in a synchronous fashion

To demonstrate this capability, an end-to-end technology selection, development and deployment was required. In this case a brownfield air compression machine was selected and it had no capability other than dispensing air. To enable this asset to speak for itself, it was equipped with various sensors for vibration, temperature, humidity, air pressure and current consumption sensors attached to it. This demonstrates how an asset can be monitored – and hence servitised – thanks to data processed through a set of business logs.

The data dispensed from sensors is picked up by a gateway using LoRaWAN wireless connectivity technology that pipes data through to the cloud environment. The data is welcomed by a message queuing service and then routed to different destinations according to type. An application programming interface (API) has been developed for integration of a mobile application that is used by customers, where they can monitor their asset data and billing information, request maintenance and receive notifications. Such interaction with the end customer is an essential part of servitisation as a continuous service, rather than a one-off sell.

The API is also used to integrate an enterprise application that is used by the service provider, through which they can visualise, manage, monitor and maintain customers and data. As a vital part of servitisation, the whole logic is integrated into a CRM where SLAs are assigned to the service users. All these IoT technologies operate together to add value to the air-as-a-service model by reducing cost and time, optimising resources, and providing better customer experiences.

Figure 1 shows a high-level architecture for the servitisation demonstrator and the key components adapted for its modelling. The key elements are the compressors and sensors unit, gateway, connectivity, cloud, enterprise business applications, CRM application and mobile application.

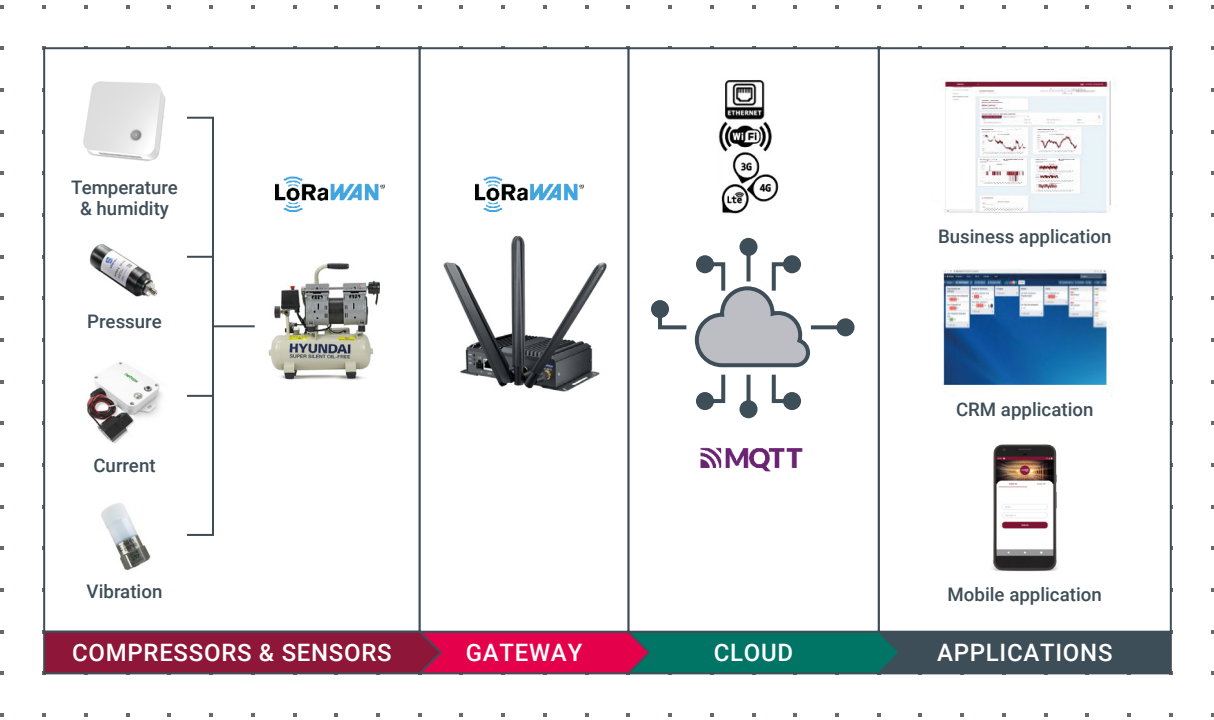


Figure 1: A high-level architecture for the servitisation demonstrator platform.

One of the goals for this demonstrator was to demonstrate that a servitisation solution can be formulated using only open source solutions, proprietary solutions or a hybrid approach – as demonstrated in this report. These combinations are interchangeable with other similar products in the market, irrespective of whether they are open source or proprietary.

The servitisation platform technology stack

A sophisticated technology stack is the backbone of a delivered solution. While keeping up-to-date with the latest technology and desiring optimal benefit from the features of services on the market, it is also important to have a flexible, ‘open box’ solution that is efficient, integrable and interoperable.

Servitisation may become complex without easy-to-integrate components – it requires a plug-and-play approach wherever possible. This is made possible with LoraWan OTA (over-the-air) technology at the sensor level, so no wiring is required, while providing an API to integrate new applications and a mobile application, to enable the various end user interactions that are vital to the business model.

The servitisation business model requires openness for integration in many technology areas, such as sensors, mobile and business application and

business intelligence (BI) tools. Selecting open source technologies over commercial off-the-shelf solutions has some benefits while bearing some risks, such as maintenance and security management (which can be addressed by a team’s experts when the right components are used in the right place). The cost benefit of open source is arguable, however it provides many advantages that are important, through upfront CAPEX, integration, interoperability and flexibility.

An agile methodology is undoubtedly the most desirable approach for solution development, especially in environments where rapid development is required for a component integration. As an example, various data formats may be added to the datapool that is then processed to servitise a value for an end customer. In such a scenario, an unsupported data protocol would need rapid development, and the integration of a module that handles operation.

Multiple elements are required to form the technology stack for a servitisation platform, as shown in Figure 2.

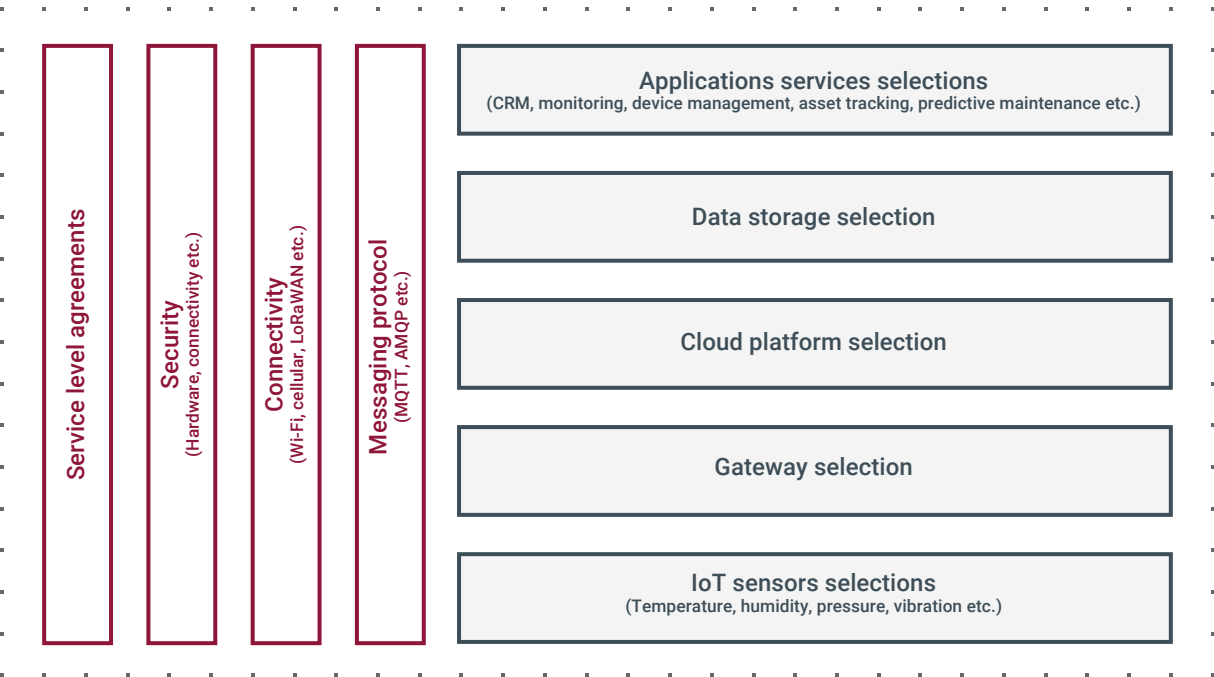
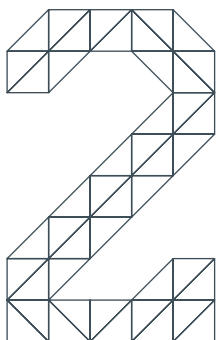
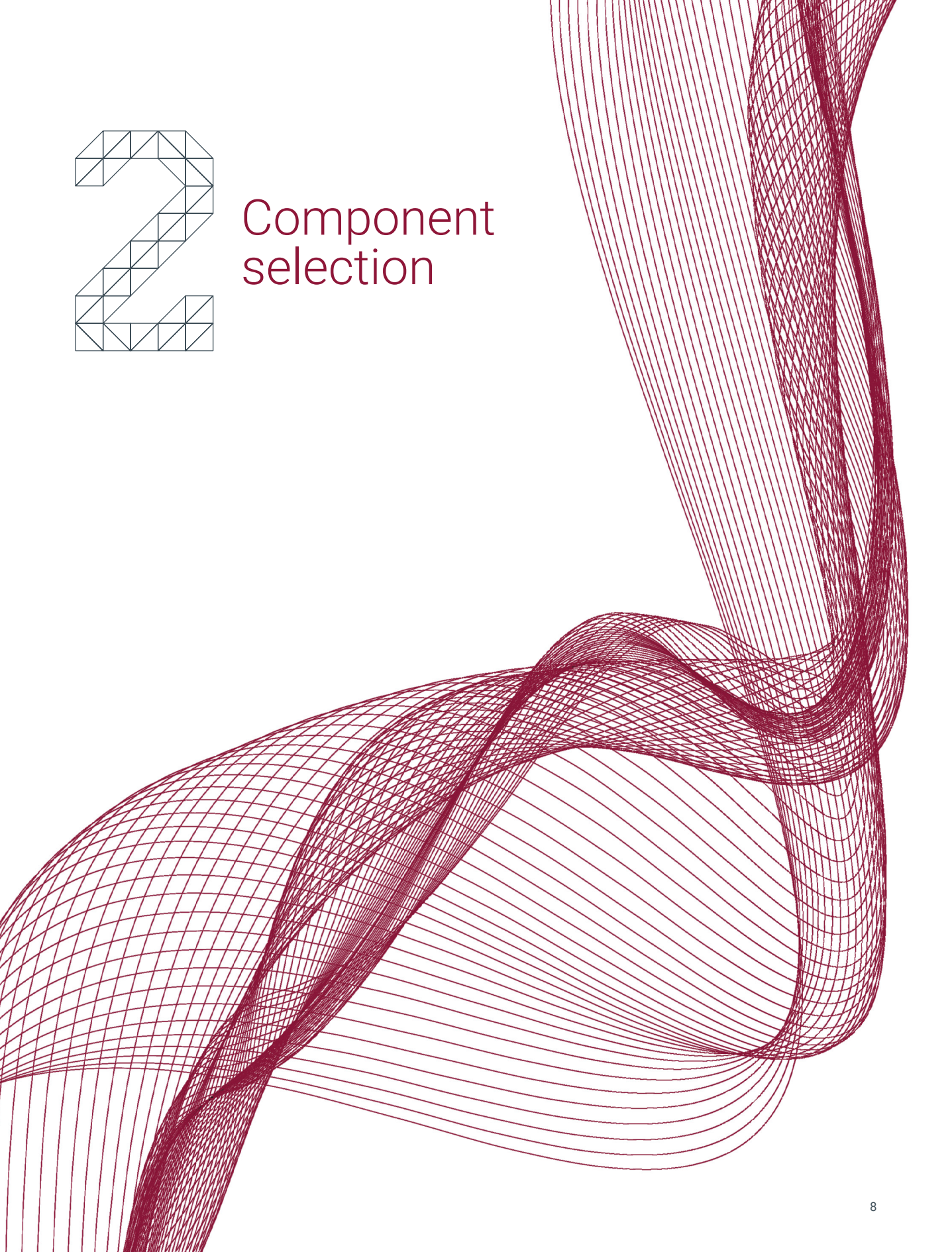


Figure 2: Technology stack needed to create a servitisation platform



Component selection



Component selection roadmap and criteria

The technology selection roadmap in Figure 3 shows the step-by-step sequence for developing a servitisation platform solution.

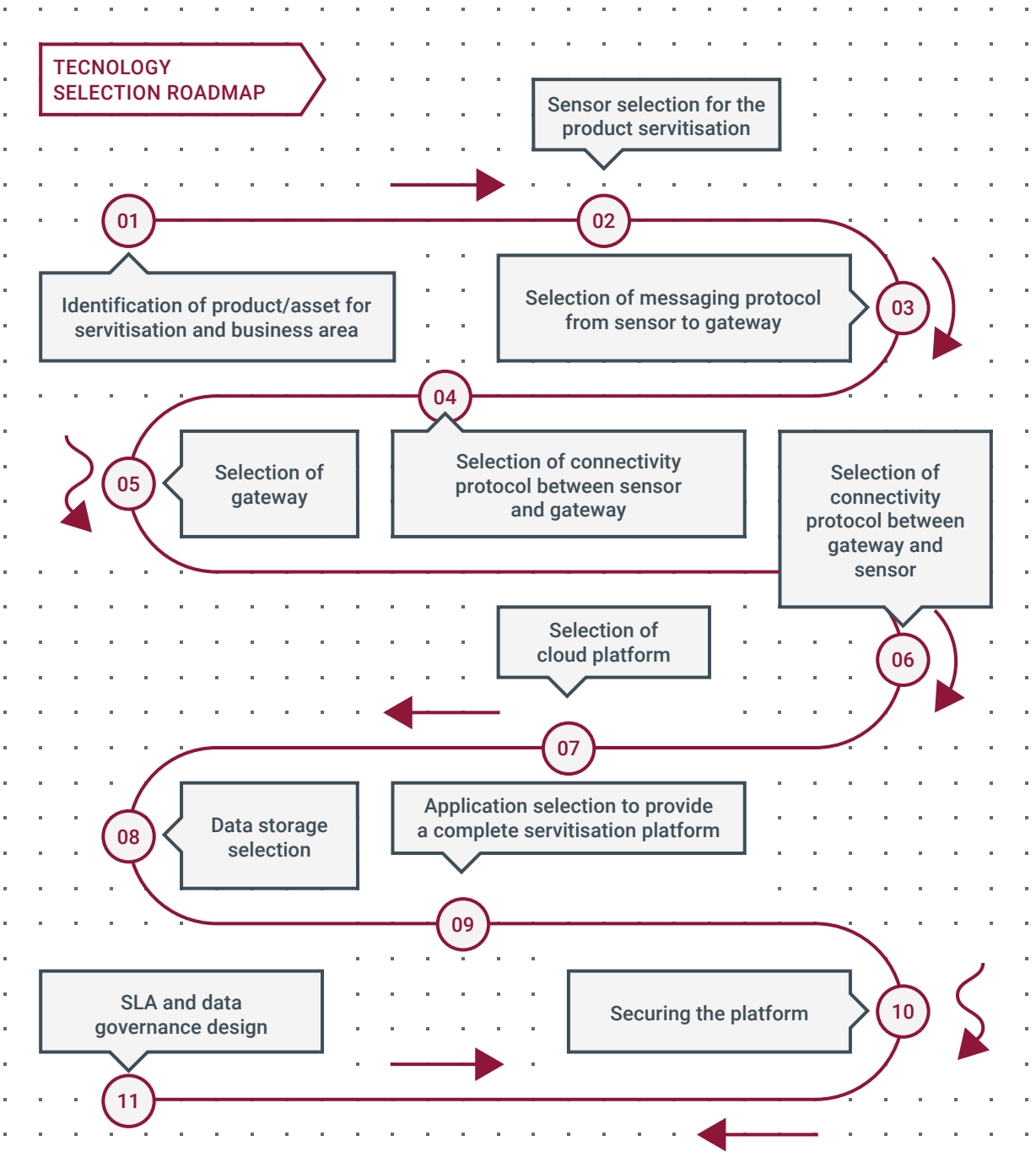


Figure 3: Technology selection roadmap

IDENTIFICATION OF PRODUCT/ASSET FOR SERVICISATION AND BUSINESS STRATEGY

This focuses on the current technology readiness level and various aspects to choose (and prioritise) such as inventory management, promotion and cross-selling, marketing automation, data governance enterprise integration, service management, and security. Businesses should clearly identify their servitisation business model with respect to the customer perspective and where the value lies in contrast to the business strategy.

SENSOR SELECTION FOR PRODUCT SERVICISATION

This depends on identifying the correct data needed for servitisation model. For a brown field asset, evaluation has to be done and there is greater freedom of shaping the servitisation model to best fit the business strategy. For a green field asset, which has some sensors attached to it and is producing data feeds then evaluation lies in enhancing these capabilities. Adding more sensors to generate more relevant and insightful data from the asset should enhance or add to existing services.

The type of sensor selected will vary based on:

- the type of data needed to be extracted from an asset and the objectives to be achieved
- the accuracy of the sensors, and how this may be affected by various factors, such as weather conditions. Accuracy may have an impact on the cost of sensors and therefore the rate of return
- the size, power consumption and shape of the sensor

Other factors include whether they are wired or wireless, the type of application running on the sensors, administration of these applications, security factors and embedded intelligence.

SELECTION OF MESSAGING PROTOCOL FOR SENSOR TO GATEWAY

Choosing the right data messaging protocol for transmission of data from the sensors to the gateway will enhance the performance (availability, data transmission rate and so on) of communication between the sensors and the gateway. Protocols include MQTT, CoAP, OPC-UA, HTTP and AMQP each is best suited to a specific application area and situation.

Selection of a standard and effective messaging protocol depends on the nature of the servitisation platform and its messaging requirements. Messaging protocols include MQTT (message queuing telemetry transport), constrained application protocol (COAP), advanced message queuing protocol (AMQP) and HTTP. Considerations include:

- encoding and decoding format
- bandwidth and latency
- interoperability
- security
- power consumption and resource requirements
- licensing model such as open source, free, or licensed
- quality of service/reliability
- transport protocol such as TCP, UDP, SCTP
- semantics
- message size and overhead
- abstraction criteria
- standards such as OASIS, IETF, ISO, W3C
- some of the other criteria are default ports, cache and proxy support

SELECTION OF CONNECTIVITY PROTOCOL FOR SENSOR TO GATEWAY

Choosing the right network protocol will result in better data transmission and efficiency. The faster the transmission rate, the quicker the service models and analytics on the edge gateway or cloud can act, which can be crucial for mission-critical applications. The right choice of protocol will also reduce running costs, improve efficiency and support advanced value added services for the asset.

Selecting the right connectivity protocol/network is important for connecting the servitisation platform components and providing optimised and high-performing services. Some of the considerations for choosing connectivity type (such as Wi-Fi, satellite, cellular, LoRaWan, NB-IoT, Bluetooth) are:

- cost
- scalability
- deployment area
- power consumption
- coverage range
- bandwidth

SELECTION OF GATEWAY

Evaluation of the factory environment and business needs is required to analyse the type of gateway and edge device necessary. Many factors have to be taken into account, including the flexibility of the architecture, hardware cost, cloud technology, protocol compatibility, and type of applications running on the gateway. Making the right selections based on combined criteria will increase the efficiency, involve new technologies and lower the costs of the servitisation model.

Criteria that should be kept in mind when selecting an edge device include:

- network connectivity and data messaging protocols
- configuration of the edge device to better manage and monitor a single or multiple edge devices
- data pre-processing capabilities of the edge device – data filtering, consolidation and rules engine. Others to be considered include data pre-analytics, local database, notifications, data compression, encryption, and the ability to track and trace data sources
- computation power of the edge device
- running open-source and third party applications
- power supply for the edge device
- backup connectivity, storage and power supply for emergencies
- security

There could also be a fog layer (another layer of edge device with higher computation abilities than the gateway layer) between the gateway and cloud.

SELECTION OF CONNECTIVITY PROTOCOL FOR GATEWAY TO CLOUD

The choice of network protocol for connecting the gateway to the cloud infrastructure plays an important part in ensuring the success of the servitisation model. The parameters to be taken into consideration are continuous availability, range, latency, power consumption, bandwidth and ease of set-up. The right choice of protocol will also reduce running costs.

SELECTION OF CLOUD PLATFORM

The right platform will support better service management and the creation of added value services.

Choosing the right cloud platform is another vital step and should take into account enterprise integration, pricing model, size of the business, and reliability, as well as the following considerations:

- public, such as open source cloud solutions; private on-premises cloud services; or a hybrid approach
- the architecture of the servitisation platform aligning with the business model and future strategy, to evaluate and avoid unnecessary expenditure in future
- compliance with standards and data regulations, such as General Data Protection Regulation (GDPR).
- cloud security
- managing the cloud and the infrastructure
- service level agreements for availability, response time, capacity, support etc. to monitor the KPIs.
- cost of operation, as well as any related hardware and software cost
- container capabilities
- scalability and flexibility to align it with changes, such as a new sales strategy or innovations
- feasibility with current IT Infrastructure.
- any requirement for data storage
- running analytics and integration of third-party applications such as CRM

SELECTION OF DATA STORAGE

Data storage is an important element of data governance. The type and nature of the storage has implications in terms of cost, complexity, and operation – examples include relational databases, time-series databases and network databases. The right data storage is important for services that rely on historical data, such as predictive maintenance.

Data storage selection will depend on the type of data being generated in the servitisation platform. This needs a careful evaluation before a selection is made, keeping relevant criteria – including future scalability and data portability – in mind.

Example criteria include:

- online storage such as public/private/hybrid or offline storage
- type of data structure structure, unstructured or hybrid
- security standards and compliance certifications
- total cost of ownership for the storage
- any license requirements
- SLAs for availability, response time, capacity, support and so on, and the ability to monitor KPIs
- capacity and scalability
- connectivity, robustness, portability and stability
- DevOps: especially the programming language skill set required by in-house IT support teams to run the infrastructure

The actual criteria will depend on the application and architecture of the servitisation model.

APPLICATIONS SELECTION TO PROVIDE A COMPLETE SERVITISATION PLATFORM

There are two types of application that may be needed: those that will provide the fundamentals of servitisation (such as CRM) and those that will improve the capabilities of an established servitisation platform to enhance and add value to the servitisation model.

Examples of services that can be incorporated include customer relationship management tools that can also be utilised for in-person field services, real-time monitoring of services on devices, customer usage evaluation models, SLA management, API management, database management, product provisioning, device/fleet management, predictive maintenance, remote diagnostics, log management services, notification services, inventory management services, asset tracking services, security services, and enterprise integration services.

SECURING THE PLATFORM

After the servitisation platform barebone is in place, the various aspects of the platform can be secured, such as the gateway, connectivity, and cloud. The reliability and integrity of servitisation model are essential to customer satisfaction.

Security and privacy in the servitisation platform should be built in from ground up. These are some of the areas to be considered when designing the platform's security.

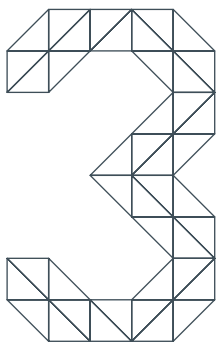
- customer and business user security through IAM policies, certificates and authentication tokens
- security of sensors, edge devices and cloud hardware through measures such as device IAM policies and certificates
- securing the connectivity and messaging protocol, such as TLS v3
- securing the database
- securing the cloud
- security policy management
- software and firmware security or implementation faults
- hardware security
- API security
- container security
- data anonymization, where required.
- access control for business users as well as customers

SLA AND DATA GOVERNANCE DESIGN

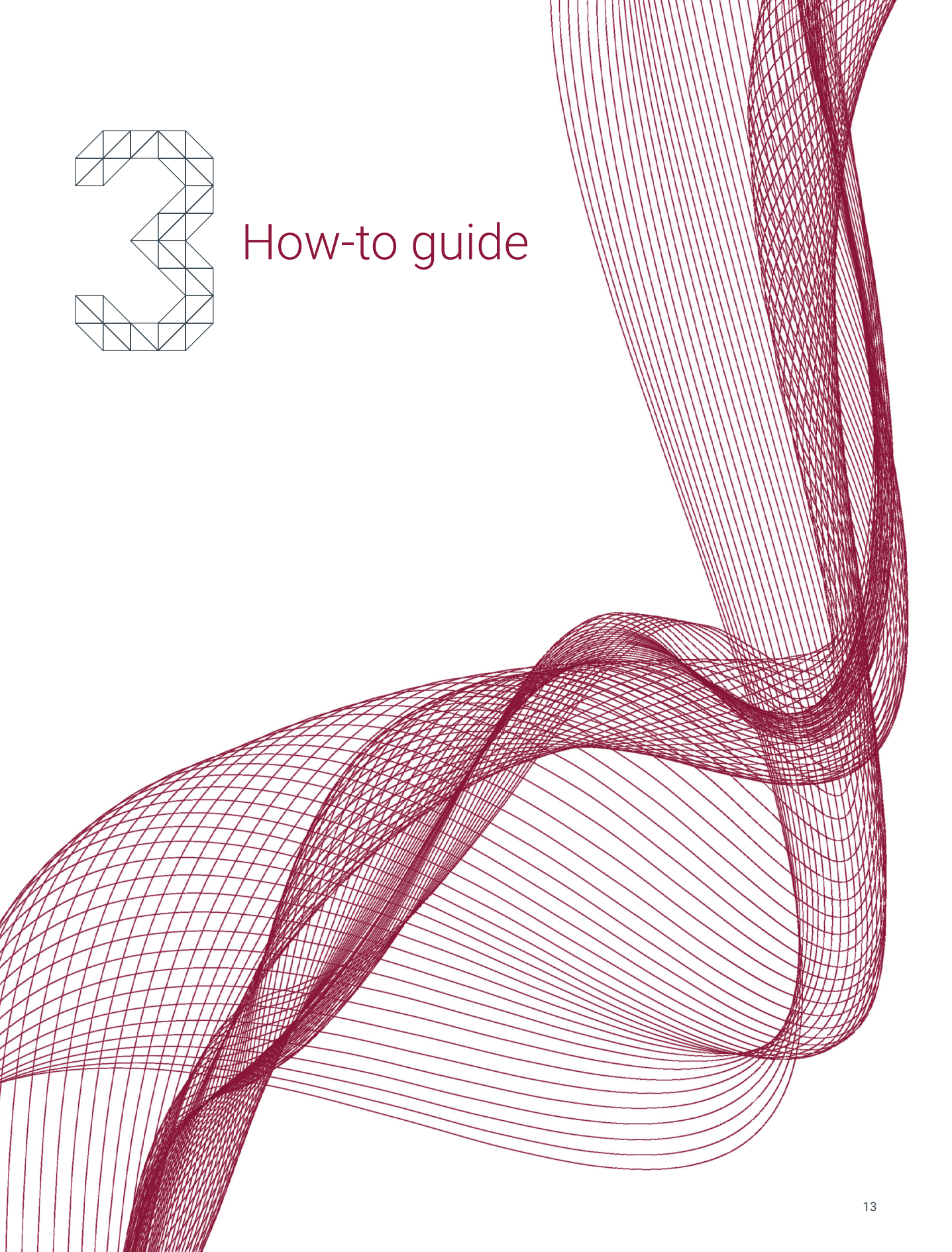
After securing the platform, one of the most vital aspects of servitisation is the formulation and modelling of SLAs, various data governance rules and network policies for the entire platform. This is vital for customer loyalty, cross-selling and service management, and keeps the servitisation model tightly knit.

A robust SLA template for inter- and intra-company services will cover:

- service performance commitment, acceptable downtime, back-off period, up-time and so on
- billing model
- service credit
- services covered



How-to guide



Servitisation demonstrator components

The key components involved in the development of the servitisation demonstrator are as listed below:

Hardware components

- Hyundai HY5508 compressor
- TE 8911-E vibration sensor
- Netvox -75A clamp-on 1-phase current meter sensor
- Ellenex pressure sensor
- Advantech WISE-6610 LoRaWAN gateway
- ERS temperature and humidity sensor
- Ethernet cable

Services components

- AWS IoT Core
- AWS Lambda services
- AWS Cloud Watch
- AWS Elasticsearch database
- AWS PostgreSQL database
- AWS serverless API
- Trello

Software components

- Visual Studio IDE
- OS- Android OS v.5.0 and Windows 10
- Postman

Data protocol components

- MQTT

Network protocol components

- LoRaWAN

Programming language components

- Python
- JavaScript
- Java
- HTML5
- CSS

Container services components

- Docker

Demonstrator component details

HARDWARE

Hyundai HY5508 compressor

We are using Hyundai air compressors of up to seven PSI rating to generate air pressure for cooling or pneumatic applications as the compressor of choice for this demonstration. We can use a similar principle to monitor heat pumps, motors, or other compressors for various applications.

The compressor is built with automatic pressure detection and safety. As soon as the pressure exceeds seven PSI it switches off, and when the pressure goes below four PSI it automatically turns on, when connected to the mains. The installation user manual gives step-by-step guidance to installing the compressor and getting it up and running.

The compressor has been purchased with accessories that are crucial to make it work.

1. Hose couplings
2. Y splitter
3. Hose coupler and tyre inflator
4. Various other optional adapters and fittings

These fittings are where pressure sensors can be added. The external Y coupler can be thread mounted using a coupler/adaptor with a push-pull mechanism on the compressor outlet hose. One of the coupler ends can be fitted with a sensor and the other fitted to the tyre inflator. Figure 4 shows the Hyundai HY5508 compressor and peripherals that were used in the design of this demonstrator.



Figure 4: The Hyundai HY5508 Compressor with attached accessories

TE 8911-E vibration sensor

TE 8911-E is used to monitor vibration at a frequency range of 1Hz to 10kHz across three axes, and is suited to industrial applications. It operates within a temperature range -20°C to 60°C.

The sensor can be stud-mounted onto the motor, or stuck on with an adhesive pad. It is battery-operated and can be turned on by sliding and turning the top white cap and pressing the button. This can also be tested with a mini-USB power cable. The user manual provides step-by-step guidance to installation and getting it up and running, and a detailed guide to the payload formatter. Figure 5 shows the TE Vibration Sensor 8911-E.



Figure 5: TE vibration sensor 8911-E

Netvox -75A clamp-on 1-phase current meter sensor

The Netvox-75A clamp-on 1-phase current meter sensor is used for measuring current, and is installed on the side of the compressor. It was chosen for its compatibility with the LoRaWAN protocol and can detect the input current of the single-phase alternating current.

To enable the device, move the magnet over the side of the sensor to turn it on. Ensure there is sufficient battery charge within the device. Figure 6 shows the Netvox -75A clamp-on 1-phase current meter sensor.



Figure 6: Netvox -75A clamp-on 1-phase current meter sensor

Ellenex pressure sensor

The Ellenex pressure sensor PTS2 is used to monitor the pressure data. This can measure any fluid or gas pressure up to 10 PSI, which should be ideal for this application (and for greater pressure, the supplier has different models). This sensor was chosen for its compatibility with the LoRaWAN protocol and high accuracy and ultra low power consumption.

To enable the sensor, pressing the button on top of the sensor with an antenna mounted to the sensor will help send the data. Ensure that the device battery has enough power.

The device setup and integration guide, with payload format information, shows a step-by-step guide to get it up and running. Figure 7 shows Ellenex pressure sensor.



Figure 7: Ellenex PSTS2 pressure sensor

Advantech WISE-6610 LoRaWAN gateway

The Advantech WISE-6610 gateway has been used for this demonstrator implementation. The gateway setup is complete and requires no accessories, apart from an Ethernet cable for internet connectivity. There are detailed step-by-step instructions of gateway setup in Section 3 of this document. Figure 8 shows the Advantech WISE-6610 gateway.



Figure 8: Advantech WISE-6610 gateway

Ethernet/RJ45 cable

An Ethernet/RJ45 cable needs to be attached to the gateway for internet connectivity. Figure 9 shows a 2-metre Ethernet cable, but the length will vary depending on the site where the gateway is to be installed.



Figure 9: Ethernet cable

ERS Temperature and humidity sensor

An Elsys sensor is used for measuring temperature and humidity over LoRaWAN protocol. This is a wireless module and powered by two 3.6V AA lithium batteries which makes it suitable for this demonstrator. To enable the sensor, press the circle button on the sensor and ensure the battery has sufficient charge. Follow this <https://www.elsys.se/en/wp-content/uploads/sites/7/2018/06/ERS-data-sheet.pdf> for a detailed specification of the sensor and for any troubleshooting. To add this sensor data to The Things Network, a payload decoder is being supplied by the supplier website. Figure 10 shows the Elsys sensor.



Figure 10: Elsys sensor

SERVICES

AWS Lambda

AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers. It can trigger Lambda from over 200 AWS services and software-as-a-service (SaaS) applications, and you only pay for what you use. We chose this solution to deploy the code for a number of reasons, including scalability, minimising provisioning and managing infrastructure.

AWS CloudWatch

Amazon CloudWatch is a monitoring and observability service. There are a number of reasons for choosing Cloudwatch.

- It provides data and actionable insights to monitor applications, respond to system-wide performance changes, and optimise resource utilisation.
- It collects monitoring and operational data in the form of logs, metrics, and events.
- It gives a unified view of operational health and has complete visibility of all the AWS resources, applications and services running on AWS and on-premises.
- It is also used to detect anomalous behaviour in your environments, set alarms, visualise logs and metrics side by side, take automated actions, troubleshoot issues, and discover insights to keep applications running smoothly.

AWS Elasticsearch database

Elasticsearch offers features to help store, manage and search time-series data, such as logs and metrics. Amazon Elasticsearch is a managed service that makes it easy to deploy, operate, and scale in the AWS Cloud. It provides all the resources for the cluster and launches it.

It can control access to your domain using AWS identity and access management (AWS IAM), and back up the data using automated or manual snapshots. Some of the other reasons why we choose this solution are that the service automatically detects and replaces failed Elasticsearch nodes, reducing the overhead associated with self-managed infrastructure and Elasticsearch software.

AWS PostgreSQL database

PostgreSQL is an advanced, enterprise-class open source relational database that supports both SQL (relational) and JSON (non-relational) querying. We chose this as it is a highly stable database management system, with high levels of resilience, integrity, and accuracy. PostgreSQL is used as the primary data store or data warehouse for many web, mobile, geospatial and analytics applications.

AWS Serverless API

This creates a collection of Amazon API Gateway resources and methods that can be invoked through HTTPS endpoints. These end-points are used to communicate with the CRM application, business application and mobile application. We chose firstly for the reason that it works well and is simple to use with AWS Elastic and AWS PostgreSQL databases. Secondly, it is reliable and the server auto scales with no payment required on idle time.

Trello

Trello is a web application which has been designed by our team as an open-source Customer relationship management (CRM) application. CRM is used for managing all the business relationships and interaction with customers and potential customers. We have designed a ticketing system around it for any issues that might arise while using the servitisation platform. We used this as it is open-source, and a CRM solution can be developed as per once customisation without any payment required. This also has a lot of plugins and API end-points to further integrate with other high-end solutions.

SOFTWARE

Visual Studio IDE

Visual Studio is an integrated development environment, and a feature-rich program that supports many aspects of software development. We have used this IDE for building the code and debugging our mobile application and business applications. Alternatively, Pycharm or any other IDE could be used.

Operating system (OS):

Android OS v.5.0 and Windows 10

The demonstrator mobile application was built on Android 5.0, and the business application was developed on Windows 10. These operating systems are used only for demonstration purposes, to show that the servitisation platform is OS-agnostic. These applications could be built on Ubuntu or any other suitable operating system.

Postman

Postman is an API platform. Postman simplifies each step of the API lifecycle and streamlines collaboration. We have used Postman as an API management platform for various end-points for the entire servitisation platform. For a much higher level production grade applications, Swagger UI could be used. Postman was used enough to manage the number of API end-points that were used for the project.

DATA PROTOCOL

MQTT

MQTT is an OASIS standard messaging protocol for the internet of things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. We have used it as a data transmitting protocol for sending data from the asset to the AWS Cloud database. The reason for using this over CoAP or HTTP protocol was that MQTT is standardised, unlike CoAP, and can reuse a single connection for sending multiple messages.

NETWORK PROTOCOL

LoRaWAN/3G/4G LTE/Wi-fi

LoRaWAN (long range wide area network) is an upper layer protocol that defines the network's communication and architecture. More specifically, it is a medium access control (MAC) layer protocol with some network layer components. It uses LoRa, but it specifically refers to the network and how data transmissions travel through it. We used this for communication between the asset and the gateway. We choose this protocol because it requires minimal power consumption and has a long signal range. For connectivity between the gateway and the internet, we have used an Ethernet cable. Apart from this, cellular connectivity such as 3G/4G LTE can also be used for connectivity between the gateway and the internet.

PROGRAMMING LANGUAGE

Python

Python is a high-level object-orientated programming language. It is widely used in development of back-end code and front-end web applications. We have used Python to develop the front end and the backend of the business application together with Flask (microframeworks for backend) and Bootstrap (front-end).

JavaScript

JavaScript is a text-based programming language used client-side and server-side to make web-pages interactive. This was used in the development of the business application.

Java

Java is also a high-level object-orientated programming language. This was used in the development of the Android application on the mobile platform.

HTML5

HTML5 (HyperText Markup Language 5) is a mark-up language used for structuring and presenting content on the world wide web. This was used to develop the business application.

CSS

Cascading style sheets (CSS) language is written in a mark-up language and is a simple mechanism for adding style (e.g., fonts, colours, spacing) to the business application web pages. This was used to make the business application visually more appealing.

CONTAINER SERVICES

Docker

Docker is a set of platform as a service products that use OS-level virtualisation to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. Docker is an open source platform for developing, shipping and running applications. We have used Docker to package the business applications and push them to the AWS Lambda services. We have used Docker as a container platform for faster deployment, mobility, repeatability and automation.

Servitisation demonstrator architecture and step-by-step guide

Figure 11 shows the detailed architecture of the servitisation demonstrator. The Made Smarter air compressor section consists of the Hyundai HY5508 compressor, TE 8911-E vibration sensor, Netvox -75A current meter sensor, Ellenex PTS2 pressure sensor, and the Elsys temperature and humidity sensor. The gateway communication consists of the Advantech WISE-6610 gateway running on LoRaWAN.

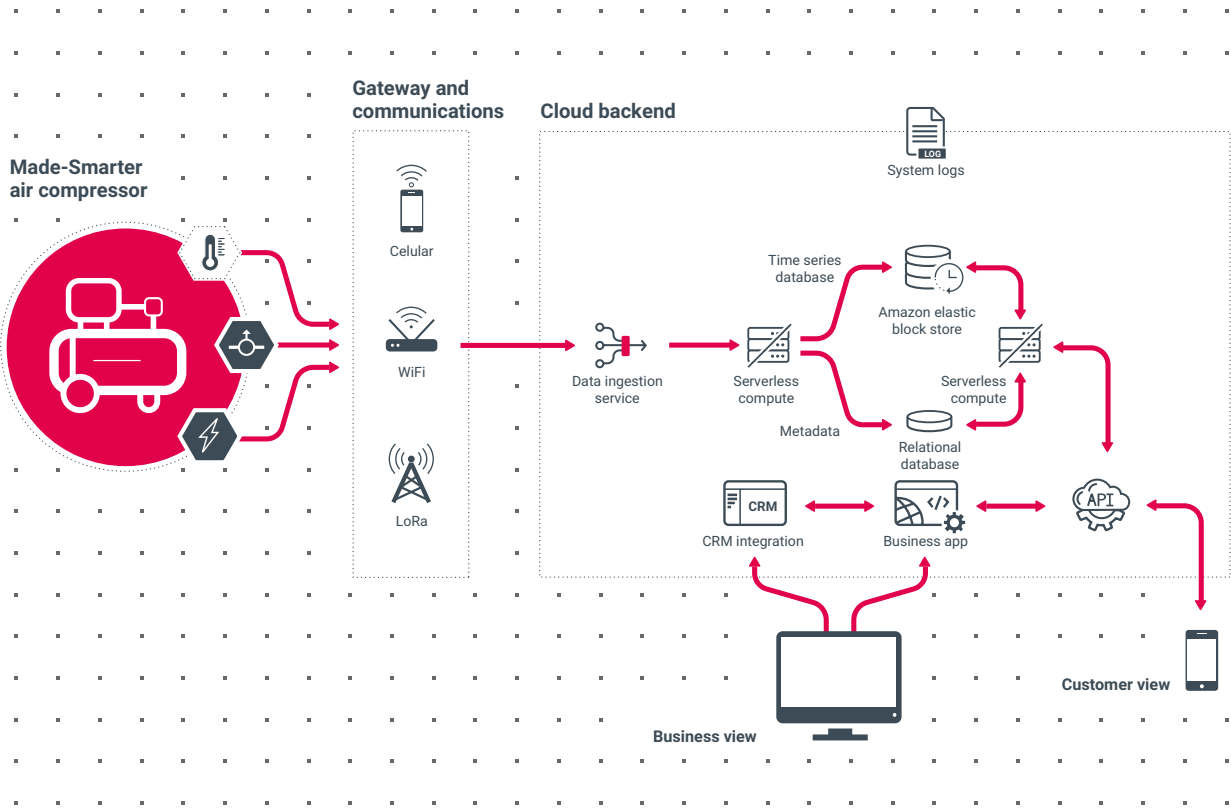


Figure 11: Detailed architecture of the servitisation demonstrator

STEP 1

Attach the respective sensors on the Hyundai compressor as shown in Figure 12, and follow the manual for each sensor to attach them to the compressor.



Figure 12: The various sensor attachments to the Hyundai compressor

STEP 2

Switch on the Advantech WISE-6610 and use the Ethernet cable to connect to your on-premises router, or to your premises Ethernet socket to connect to the internet.

Get the IP address of the gateway and access the user interface (UI) through your browser. Login using the credentials provided.

Select the right LoRaWan frequency and direct the gateway to the TTN network server endpoint.

Navigation

Router

LoRaWAN Radio

• Packet Forward

• LoRaWAN Status

Network Server

MQTT

Application Server

Licenses

Return to Router

LoRaWAN Gateway Settings

Basic Status

Data Record Time : 2018-07-10T16:15:11Z

Total Up Stream : 185 Bytes

CRC OK packet : 2

CRC Bad packet : 785

NO CRC packet : 0

Channel Status

Channel	Radio Index	Enabled	Frequency(Hz)	Received(Bytes)
0	0	Enabled	902300000	21
1	0	Enabled	902500000	84
2	0	Enabled	902700000	21
3	0	Enabled	902900000	0
4	1	Enabled	903100000	42
5	1	Enabled	903300000	17
6	1	Enabled	903500000	0
7	1	Enabled	903700000	0
std	0	Enabled	903000000	0

Last Up Stream

Index Data

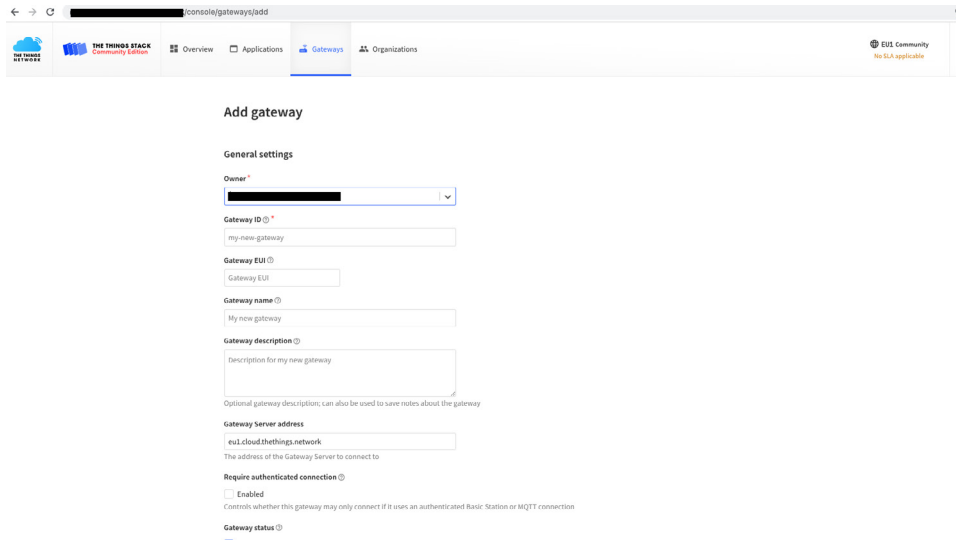
```
1 { "rxpk": [{" "tmst":3066556676,"time":"2018-07-10T11:30:00,6068981Z","chan":1,"rfch":0,"freq":902.500000,"stat":1,"modu":"LORA","datr":"SF10BW125","codr":4/5,"tsnr":-14.0,"rssl":-109,"size":
2 { "rxpk": [{" "tmst":3135274452,"time":"2018-07-10T11:31:09,322687Z","chan":1,"rfch":0,"freq":902.500000,"stat":1,"modu":"LORA","datr":"SF10BW125","codr":4/5,"tsnr":-12.5,"rssl":-107,"size":
3 { "rxpk": [{" "tmst":4189579524,"time":"2018-07-10T11:48:43,651567Z","chan":4,"rfch":1,"freq":903.100000,"stat":1,"modu":"LORA","datr":"SF10BW125","codr":4/5,"tsnr":-10.8,"rssl":-112,"size":
4 { "rxpk": [{" "tmst":2200128396,"time":"2018-07-10T13:38:44,176042Z","chan":1,"rfch":0,"freq":902.500000,"stat":1,"modu":"LORA","datr":"SF10BW125","codr":4/5,"tsnr":-11.2,"rssl":-107,"size":
```

Figure 13: GUI screen for Advantech WISE-6610 gateway configuration

STEP 3

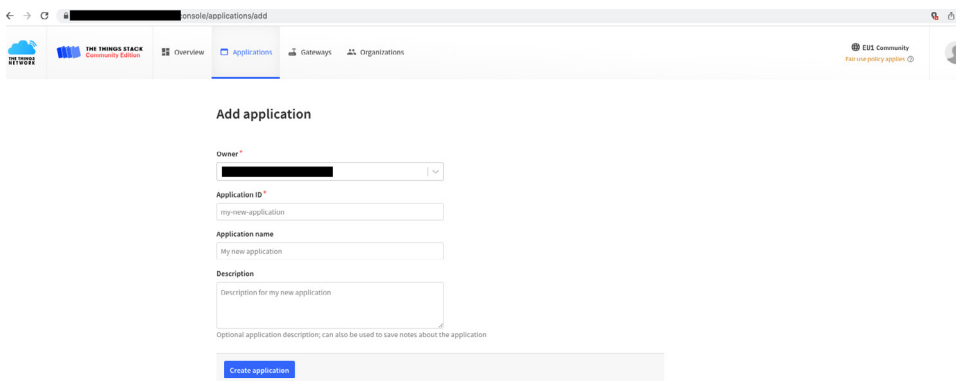
Add the gateway to The Things Network (TTN) as shown in Figure 14, using a gateway ID name of your choice, your frequency plan, and any delays to be configured on the gateway. The rest of the information is auto-filled by the The Things Network stack gateway.

Then go to the Applications tab to add the application required to onboard the sensors to the server, as shown in Figure 15.



The screenshot shows the 'Add gateway' form in the TTN console. The form is titled 'Add gateway' and is located under the 'Gateways' tab. It contains several sections: 'General settings' with fields for 'Owner' (a dropdown menu), 'Gateway ID' (a text input field with the value 'my-new-gateway'), 'Gateway EUI' (a text input field with the value 'Gateway EUI'), 'Gateway name' (a text input field with the value 'My new gateway'), and 'Gateway description' (a text area with the value 'Description for my new gateway'). Below these fields is a note: 'Optional gateway description; can also be used to save notes about the gateway'. There is also a 'Gateway Server address' field with the value 'eu1.cloud.things.network' and a note: 'The address of the Gateway Server to connect to'. A 'Require authenticated connection' section has a checkbox labeled 'Enabled' which is currently unchecked, with a note: 'Controls whether this gateway may only connect if it uses an authenticated Basic, Station or MQTT connection'. At the bottom, there is a 'Gateway status' section with a dropdown menu.

Figure 14: Adding a gateway to the TTN



The screenshot shows the 'Add application' form in the TTN console. The form is titled 'Add application' and is located under the 'Applications' tab. It contains several sections: 'Owner' (a dropdown menu), 'Application ID' (a text input field with the value 'my-new-application'), 'Application name' (a text input field with the value 'My new application'), and 'Description' (a text area with the value 'Description for my new application'). Below these fields is a note: 'Optional application description; can also be used to save notes about the application'. At the bottom of the form is a blue button labeled 'Create application'.

Figure 15: Adding an application to the TTN

Figure 16 shows how the sensors are added into the TTN application one by one, either by searching the LoRaWAN repository for the device brand and entering the registration number for the device, or by manually adding the frequency plan, LoRaWAN version and regional parameters, generating a DevEUI, AppEUI, AppKey, and device ID for the device and then registering. The step is self-explanatory on the TTN website.

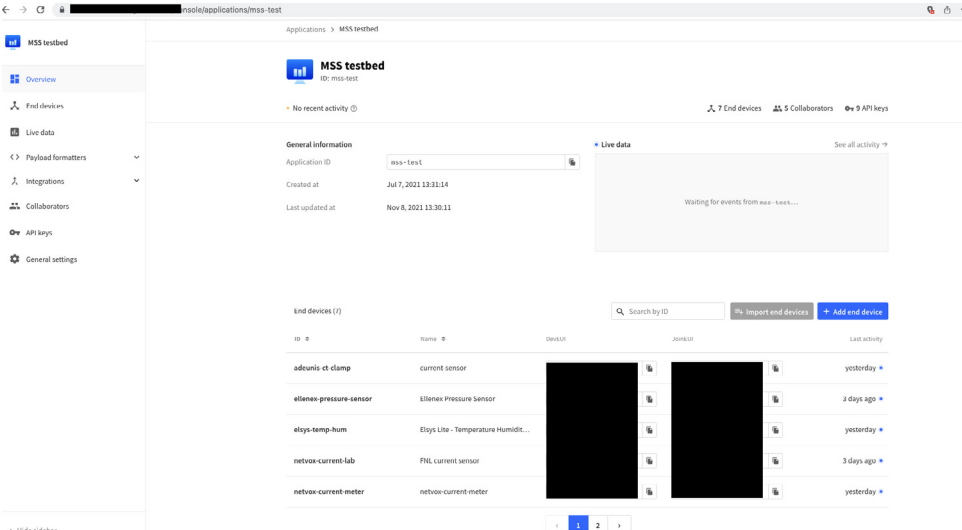
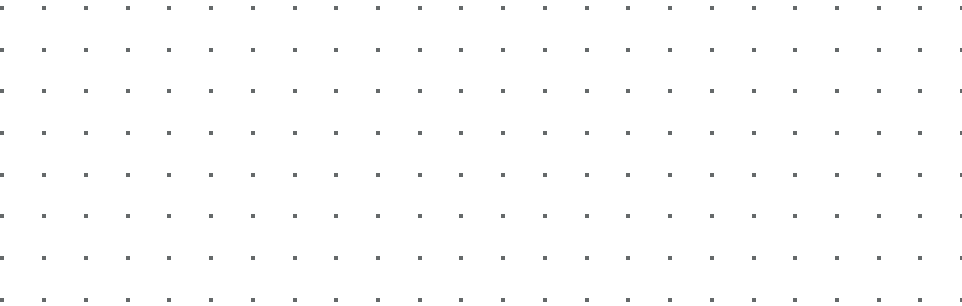


Figure 16: Adding sensors into the TTN

STEP 4

We set up the CloudWatch environment. To use Amazon CloudWatch, an AWS account is needed, allowing use of the services residing in the cloud backend environment as shown in Figure 11 (for example, Amazon EC2, AWS Lambda, PostgreSQL, Elastic Container registry).

These generate metrics that you can view in the CloudWatch console, a point-and-click web-based interface. This monitors the metrics of everything in the cloud backend, shows an aggregated view of all runtime activities on the cloud and generates logs.



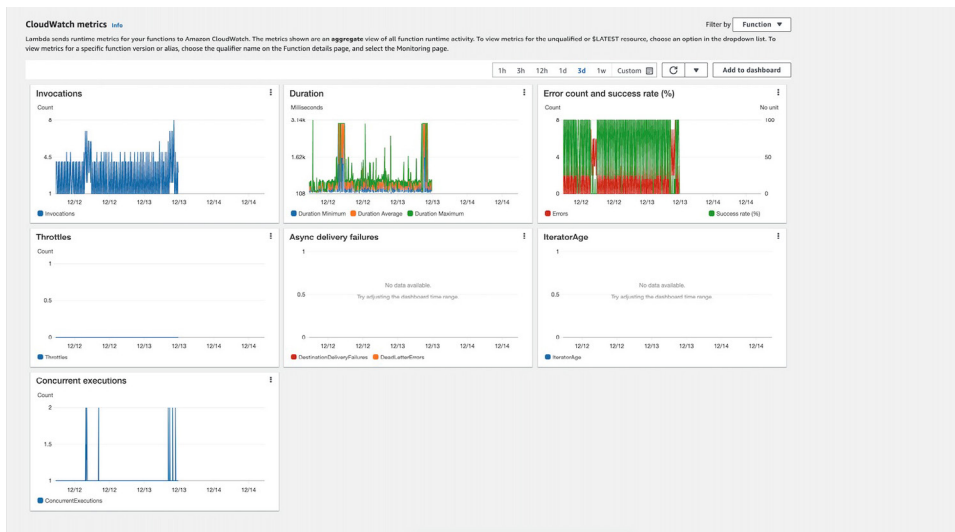


Figure 17: AWS CloudWatch metrics

Figure 18 shows a preview of the AWS Lambda environment, where all the code developed for the environment is deployed. This includes all the settings and rules for working with the TTN environment, PostgreSQL, serverless API and Elastic container.

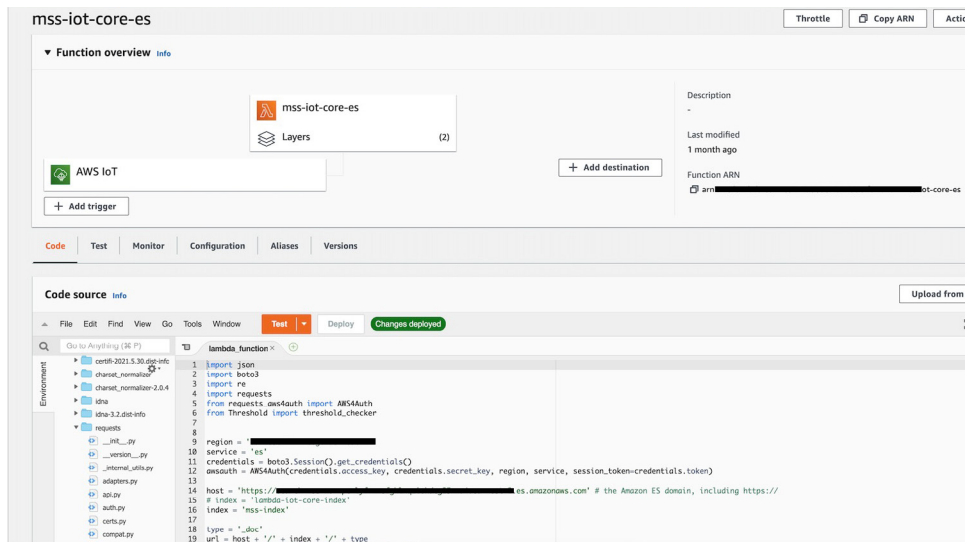


Figure 18: A view of AWS Lambda

STEP 5

In this step we set up the serverless API gateway, as shown in Figure 19. This contains the endpoints for the following:

- the login credentials for the user of the mobile application on the customer side
- notifications for the customer
- the login credentials for the user of the business application on the business side
- notifications for the business
- consumption of the asset
- asset types and details
- temperature sensor data access
- pressure sensor data access
- current sensor data access
- humidity sensor data access
- vibration sensor data access
- notifications for the asset
- registration of new customers on the mobile application
- registration of new users on the business application
- maintenance of the asset
- tariff for the specific customer

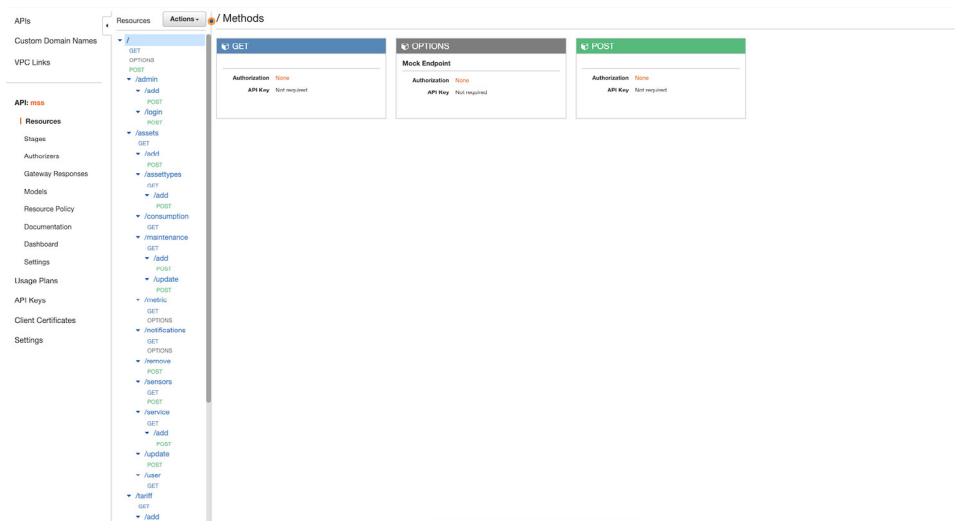


Figure 19: The serverless API gateway

STEP 6

Next we set up the PostgreSQL database with a number of tables, as shown in Figure 20.

- Access types for users of the business applications
- Admin table
- Asset table that contain list of assets and their details
- Asset metering for the tariff calculation
- Asset type for various types of asset
- Events for all events happening in the asset
- Maintenance requests being raised by the customer
- Metering type for the service the customer is using and the service level availability
- Notification type
- Sensor type
- Sensor details
- User notifications raised on the mobile application, such as user maintenance requests or asset anomalies
- User account for the customer

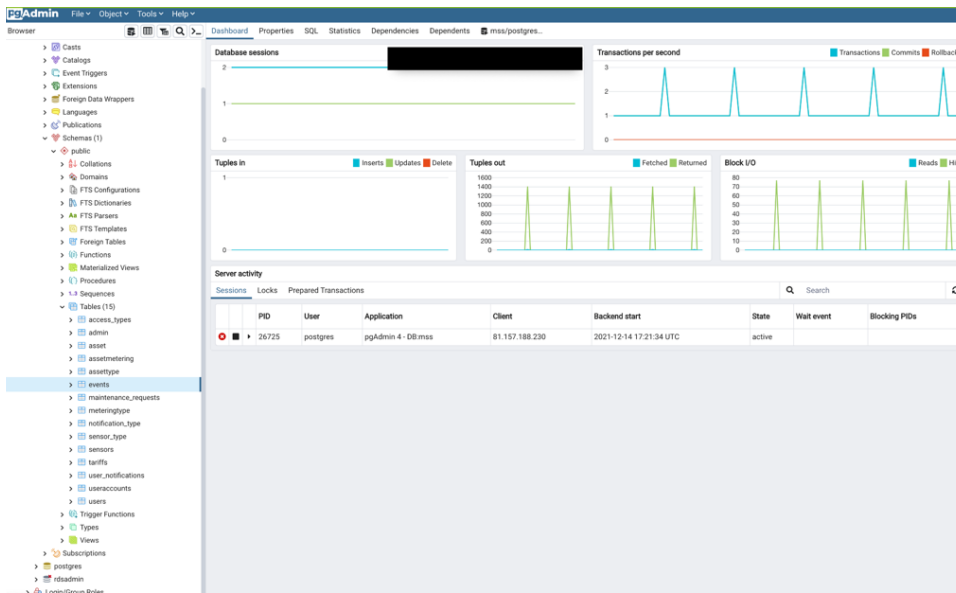


Figure 20: The PostgreSQL view of the various tables

STEP 7

The next step is to set-up the Elastic database on AWS for storing the various sensor data on the database, as shown in Figures 21 to 24.

- Figure 21 shows pressure sensor data information, including sensor ID, timestamp, payload and device EUI

- ```
Tools Dev Tools
Console
History Settings Help

115- }
116- }
117-
118 # Query with sensor IDs
119 GET mss-index/_search
120- {
121- "sort": [
122- {
123- "timestamp":{
124- "order": "asc"
125- }
126- },
127- {
128- "size": 10,
129- }
130-],
131- "query": {
132- "bool": {
133- "should": [
134- {
135- "match": {
136- "dev_eui": "[REDACTED]"
137- }
138- },
139- {
140- "match": {
141- "dev_eui": "[REDACTED]"
142- }
143- }
144-]
145- }
146- },
147- "GET mss-index/_search
148- {
149- "sort": [
150- {
151- "timestamp":{
152- "order": "desc"
153- }
154- },
155- {
156- "size": 10000,
157- }
158-],
159- "query": {
160- "terms": {
161- "dev_eui": ["[REDACTED]", "[REDACTED]", "[REDACTED]"]
162- }
163- }
164- }

10- "hits": {
11- "total": {
12- "value": 430,
13- "relation": "eq"
14- },
15- "max_score": null,
16- "hits": [
17- {
18- "_index": "mss-index",
19- "_type": "_doc",
20- "_id": "tC[REDACTED]ot",
21- "_score": null,
22- "_source": {
23- "timestamp": "2021-10-20T08:37:27.688Z0581Z",
24- "device_id": "[REDACTED]",
25- "device_name": "elimax-gpspressure-sensor",
26- "raw_payload": "A8AAAAAACI=",
27- "payload": {
28- {
29- "metric": 0,
30- "sensor_type": "pressure",
31- "unit": "bar"
32- },
33- {
34- "metric": 3.4000000000000004,
35- "sensor_type": "battery",
36- "unit": "V"
37- }
38-]
39- },
40- "sort": [
41- | 1634908247688
42-]
43- },
44- {
45- "_index": "mss-index",
46- "_type": "_doc",
47- "_id": "wC[REDACTED]p.",
48- "_score": null,
49- "_source": {
50- "timestamp": "2021-10-20T08:37:27.632Z0231Z",
```

[illegible]

## Made Smarter servitisation demonstrator

```
Dev Tools
Console
History Settings Help
1 16- 17- 18- 19- 20- 21- 22- 23- 24- 25- 26- 27- 28- 29- 30- 31- 32- 33- 34- 35- 36- 37- 38- 39- 40- 41- 42- 43- 44- 45- 46- 47- 48- 49- 50- 51- 52- 53- 54- 55- 56- 57-
##
lam
bda
-io
t-c
ore
-in
dex
###
#
2 GET
3
4 GET
5 {
6 {
 "qu
 ery
 ":
 {
 "m
 atc
 h_a
 ll"
 }
}
8 -
9 -
10
```

```
"hits": [
 {
 "_index": "lambda-iot-core-index",
 "_type": "doc",
 "_id": " ",
 "_score": null,
 "_source": {
 "@timestamp": "2021-10-19T11:15:30.557843701Z",
 "event": {
 "dev_eui": " ",
 "device_id": "elsys-temp-hum",
 "row_payload": " ",
 "payload": [
 {
 "metric": 19.3,
 "sensor_type": "temperature",
 "unit": "C"
 },
 {
 "metric": 51,
 "sensor_type": "humidity",
 "unit": "%"
 },
 {
 "metric": 3.673,
 "sensor_type": "battery",
 "unit": "V"
 }
]
 }
 },
 "sort": [
 1634642130557
]
 },
 {
 "_index": "lambda-iot-core-index",
 "_type": "doc",
 "_id": " ",
 "_score": null,
 "_source": {
 "@timestamp": "2021-10-19T11:10:30.583236012Z",
```

Figure 23: The AWS Elastic database for storing temperature and humidity sensor data

```
Dev Tools
console
History Settings Help
amp
":{
 "or
 der
 ":
 {
 "de
 sc"
 }
}
26-
27-
28-
29-
30-
31-
32-
33-
34-
35-
36-
37-
38-
39-
40-
41-
42-
43-
44-
45-
46-
47-
48-
49-
50-
51-
52-
53-
54-
55-
```

```
"_score": null,
"hits": [
 {
 "_index": "mss-index",
 "_type": "doc",
 "_id": " ",
 "_score": null,
 "_source": {
 "@timestamp": "2021-11-12T00:02:24.506641716Z",
 "dev_eui": " ",
 "device_id": "netvox-current-lab",
 "row_payload": " ",
 "payload": [
 {
 "metric": 1.3,
 "multip": 32,
 "sensor_type": "battery",
 "unit": "V"
 },
 {
 "metric": 17408,
 "sensor_type": "current",
 "unit": "mA"
 }
]
 },
 "sort": [
 1636673544506
]
 },
 {
 "_index": "mss-index",
 "_type": "doc",
 "_id": " ",
 "_score": null,
 "_source": {
 "@timestamp": "2021-11-12T00:02:24.506641716Z",
 "dev_eui": " ",
 "device_id": "netvox-current-lab",
 "row_payload": " ",
 "payload": [
```

Figure 24: The AWS Elastic database for storing current sensor data

## Mobile application development

The demonstrator includes a mobile application for use by servitisation customers. It is a native Android application coded in Java, and is supporting devices running Android OS v.5.0 or newer (API level 21). It was designed and developed taking into consideration all guidelines proposed by Google, and uses the Jetpack suite to ensure backward compatibility.

The application uses an authentication mechanism via the login page (shown in Figure 23) which can notify the user in case of wrong credentials or nonexistent accounts. After successful login, it presents a dashboard that includes usage statistics and recent incoming messages from the system.

Additionally, a user can use the menu to navigate to a screen that lists all their assets, and can retrieve detailed information about them, including historical data generated by the sensors on board each asset.

The application also runs an MQTT client through a service that enables two-way communication between the Android phone/user and the system. Based on this mechanism, an alarm and notification system alerts clients to critical usage events or maintenance scheduling. This system can be extended to support any type of communication.

Finally, the application is Firebase-ready in case another notification mechanism is needed or the need to explore in-app analytics is required.

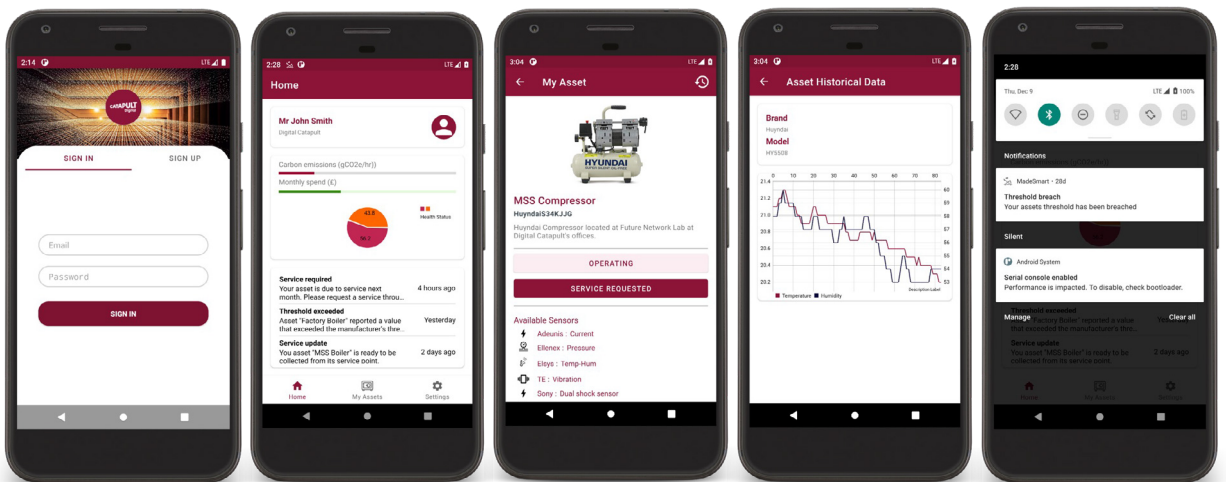


Figure 25: Mobile application features

## Business application configuration

The servitisation demonstrator includes a business application for business users, designed and developed to demonstrate some of its core features.

The backend was developed using the Bootstrap and Flask development framework, coded in Python. The frontend of the application was developed in HTML5, CSS and Javascript. Then the entire business application is containerised using Docker containers and deployed on the AWS Lambda EC2 server.

The application has an authentication mechanism via a login page (shown in Figure 26) which can notify the business user in case of wrong credentials or nonexistent accounts.

CATAPULT  
Digital

Username

Kashif Rabbani

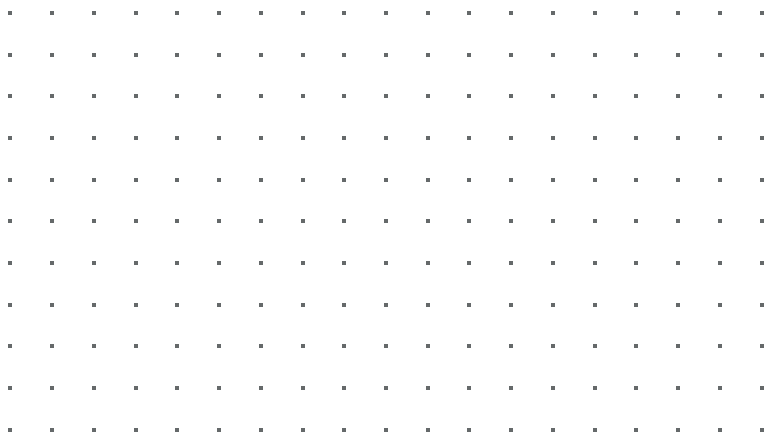
Password

\*\*\*\*\*

Login

**Figure 26:** Login page for the business application

After a successful log in, it presents a dashboard for the customer to choose from a drop-down list as shown in Figure 27. This presents an enterprise dashboard showing various metrics such as consumption of the asset for the user, temperature, pressure, humidity, voltage, vibration.





## Dashboard

[Home](#) > [Dashboard](#)

Choose an Customer:

Submit

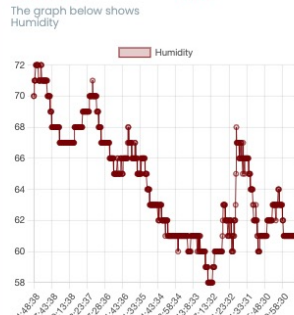
## Consumption

Aggregated consumption for all assets for a specific month



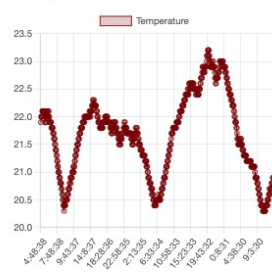
## Humidity

The graph below shows Humidity



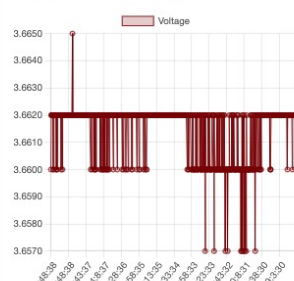
### Temperature

The graph below shows Temperature



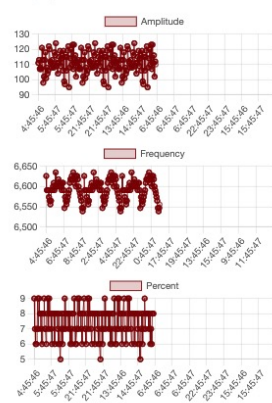
Voltage used

The graph below shows Voltage



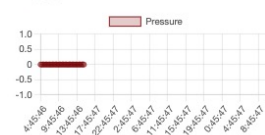
## Vibration

The graph below shows Vibration



## Pressure

The graph below shows Pressure

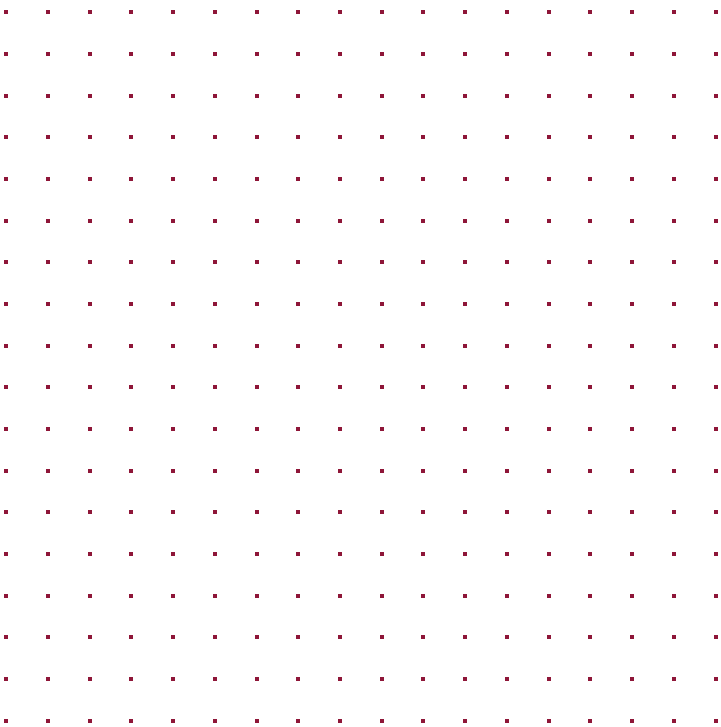


**Figure 27:** Example of the application's enterprise dashboard

Additionally, a business user can navigate using the menu on the left of the screen that lists customer profiles as shown in Figure 28. Here a customer can be chosen from the dropdown list and his credentials can be viewed. The customer details tab shows all customers currently listed on the system and currently using the assets services. If a specific customer is to be searched then it can be searched via the search bar beside the Display all user tab. This page also contains the ticketing system which is linked to the Trello CRM application (shown in Figure 31).

The CRM application is embedded into the business application using APIs. The ticketing tab can be used to raise any tickets against any maintenance request raised by the user which gets directly logged into the CRM application. The CRM application contains an SLA for each ticket to be resolved within 24 hours.

The Asset tab in Figure 28 allows to choose a customer and the specific asset which belongs to the customer from a drop down list. It presents Asset details for that customer and the list of sensors attached to that specific asset can also be viewed via the Attached List of Sensor details. Then the list of sensors can be searched to get details about a specific sensor. Furthermore, historical data generated by the sensors onboard every asset such as humidity, temperature, current voltage, vibration and pressure is displayed.



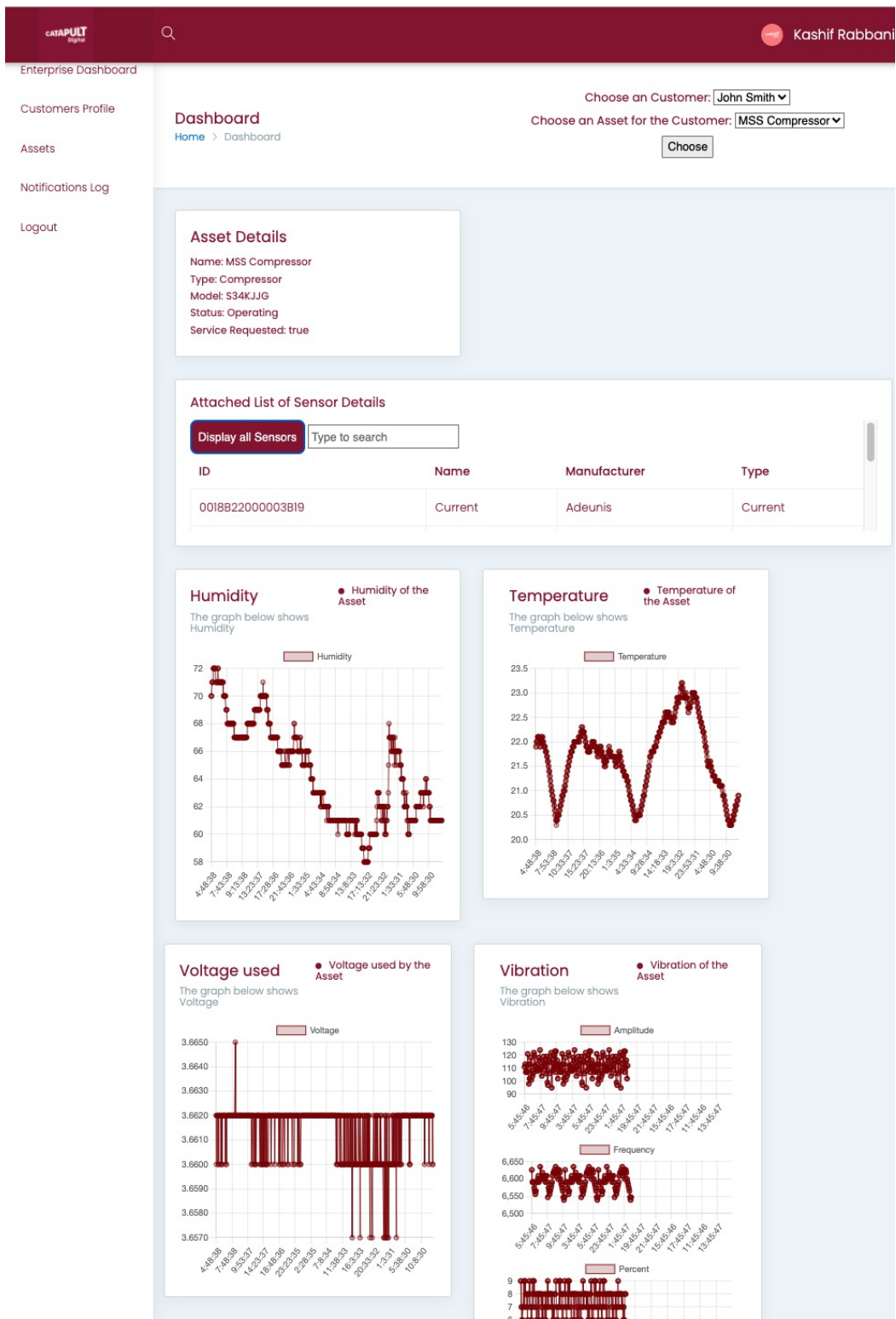


Figure 28: Example of the application's enterprise dashboard



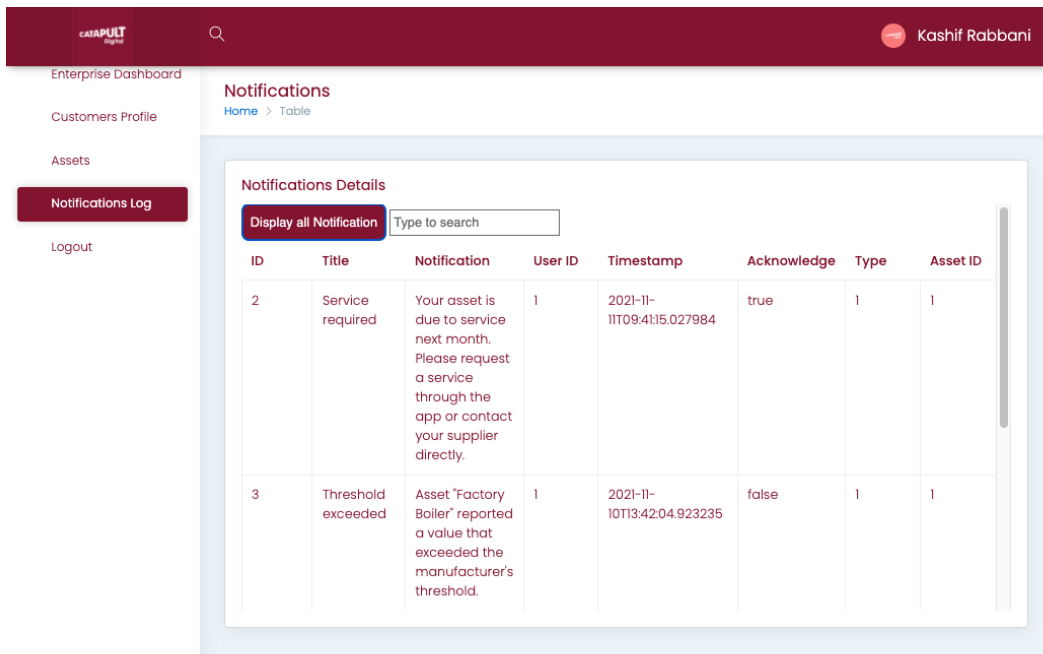


Figure 30: Notification log page for the business application

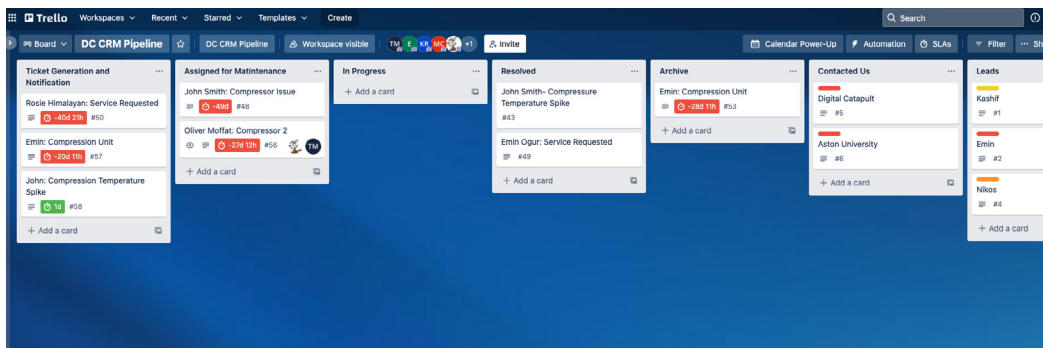


Figure 31: Trello CRM on the business application (accessible via Customer Profile tab on the left menu)



If you would like further information on Digital Catapult's work on the Made Smarter servitisation demonstrator, please visit our website.

