Design and Implementation of CI\CD over LoRaWAN

Continuous Integration and Deployment in LoRaWAN Edge Computing Applications

Smart Systems

Master's Degree Programme in Information and Communication Technology

Department of Computing, Faculty of Technology

Master of Science in Technology Thesis

Author:

El Motasim Gumaa

Supervisors:

MSc (Tech) Jorge Peña Queralta

Assoc. Prof. Tomi Westerlund

July 2022

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin Originality Check service.

Master of Science in Technology Thesis
Department of Computing, Faculty of Technology
University of Turku

Subject: Smart Systems
Programme: Master's Degree Programme in Information and Communication Technology
Author: El Motasim Gumaa
Title: Design and Implementation of CI\CD over LoRaWAN.
Number of pages: 62 pages
Date: July 2022

The recent rise of IoT devices in commercial and industrial spaces has created a demand for energy-efficient and reliable communication solutions. Communication solutions used on IoT devices vary depending on the applications. Wireless Low Power Wide Area Network (LPWAN) technologies have proven benefits, including long-range, low power, and low-cost communication alternatives for IoT devices. These benefits come at the cost of limitations, such as lower data rates. At the same time, the demand for faster, cheaper, and more reliable software deployment is becoming more critical than ever before.

This thesis aims to find a way of having an automated process where software could be remotely deployed into LoRa nodes and investigate whether it is possible to implement a DevOps pipeline with both Continuous Integration (CI) and Continuous Deployment (CD) over LoRaWAN. For this thesis, an
IoT LoRaWAN Edge computing application was chosen to determine how to design and implement a CI/CD pipeline to ensure a dependable and a continuous software deployment to the LoRaWAN nodes.
Designing and implementing a Continuous Deployment pipeline for this IoT application was made possible with the integration of DevOps tools like GitHub and a TeamCity automation server. Additionally, a series of scripts have been designed and developed for this case, including automated tests, integration to cloud services, and file fragmentation and defragmentation tools. For software deployment and verification to the LoRaWAN network, a program was designed to communicate with the LoRaWAN network server over the WebSocket communication protocol.

The implementation of DevOps in LoRaWAN applications is affected by the limitations of the LoRaWAN protocol. This thesis argues that these limitations can be eliminated using modular software and file fragmentation techniques. The implementation presented in this work can be extended for various time-critical use cases. The solution presented in this thesis also opens the door to combining LoRaWAN with other LPWAN technologies, like NB-IoT, that can
be activated on demand.

Keywords: LoRaWAN, Edge Computing, DevOps, FUOTA, Continuous Integration, Continuous Deployment

Table of contents

1 Introduction 5
1.1 Related Work 7
1.2 Thesis Contribution 8
1.3 Thesis structure 9
2 DevOps 10
2.1 What is DevOps 10
2.2 DevOps Metrics 12
2.3 DevOps process 14
2.4 DevOps Adoption Benefits 18
3 LoRa 20
3.1 What is LoRa? 20
3.2 LoRaWAN Architecture and Components 21
3.3 LoRaWAN End Device Classes 22
3.4 End Device Activation in LoRaWAN 23
3.5 End-Device Firmware Update Over The Air (FUOTA) 26
3.6 LoRa Multicast 28
3.7 Challenges of remote Firmware deployment over LoRaWAN networks 31
4 Case: CI/CD over LoRa 33
4.1 Design 33
4.2 LoRaWAN Setup 34
4.2.1 LoRaWAN Server and Gateway Setup 34
4.2.2 LoRaWAN end device setup 38
4.2.3 File Deployment over LoRa Network Server 43
4.3 CI\CD Pipeline 46
4.3.1 Face Recognition Application Development Pipeline 47
4.3.2 Images Encodings Deployment Pipeline 49
5 Results 51
5.1 Potential Use Cases and Applications 54

6 Conclusion 58

References 59

Appendices 61

Appendix 1 list of figures 61

Appendix 2 list of tables 62

1 Introduction

The global market for semiconductors has grown drastically over the past decades [1][2][3]. The usage of intelligent devices, robots and computers has grown substantially in domains like smart cities, industrial robots, and smart homes [4][5][6]. This high demand and expansion of domains have been accelerated and fuelled by the revolution of computer science, technologies, and telecommunication to the point that it has become an integral part of human lives. Those intelligent devices and computers have proven reliable, efficient, and accurate in accomplishing tasks that humans find repetitive, too complicated, and time-consuming. The increased number and complexity of these systems and the tasks delegated to them have created the need to connect these devices (Machine-to-Machine) and connect them to the Internet. The ecosystem of these "things" connected to the Internet has introduced terms like cyber-physical systems, IoT (Internet of Things) and edge computing. The complexity and quantity of these devices and the expansion of usage domains required secure and continuous updates of these devices with the latest technologies and software [7]. This continuous software development is critical to meeting information security and business requirements.
The software industry's evolution has produced plenty of technologies, practices, and methodologies in an ever-ending quest to solve emerging issues with every recent technology, software, development method or development framework.

The ever-changing nature of software under development introduced through continuously developing new features and fixing discovered bugs creates a need for innovating new, robust, and reliable ways of testing, deployments and even refining and prioritizing the tasks in the backlog. The ability to be agile with development priorities over time in response to market needs while focusing on the main development goals and improving the quality has proven crucial for the software under development and, therefore, the success of the businesses behind it.

In traditional software development, predictive methodologies like Waterfall, Sashimi and V-Model are used [8]. In the Waterfall method, the development process goes through long, isolated, sequential, and rigid phases [9]. In these isolated phases, different teams work in silos where many assumptions are made, and the customer or the stakeholder is only involved in the first and last phases.

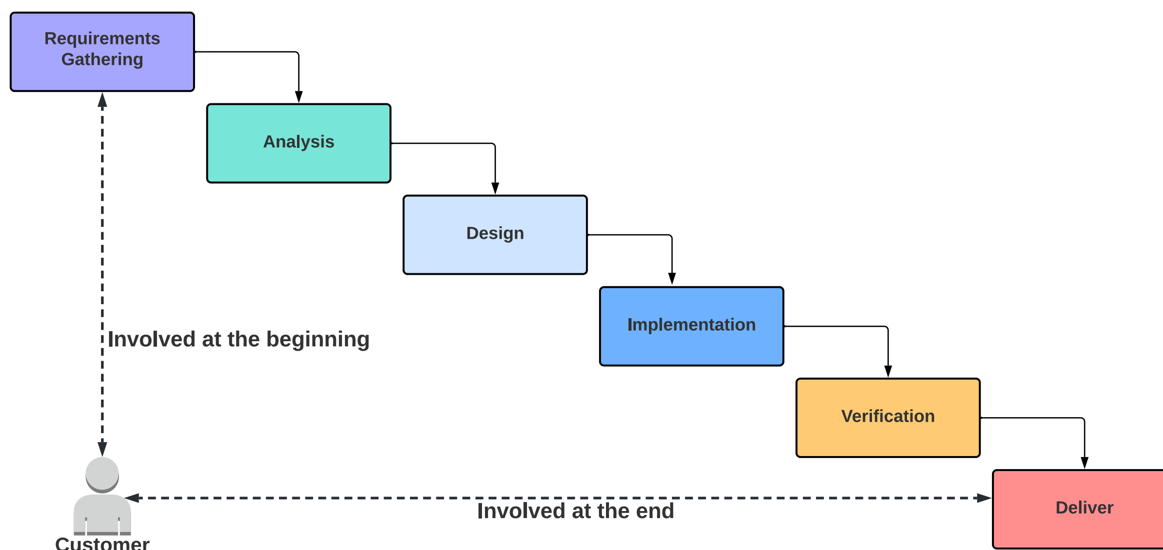

Figure 1-1 Waterfall Software Development Phases

Agile development and DevOps practices were designed to mitigate known disadvantages of conventional methods like Waterfall [9]. The Waterfall software development method has disadvantages like inflexibility and minimum customer involvement. Delays in any phase in such a method are also meant to delay the following phases of the development [8][9].

When it comes to the IoT industry evolution, there has been a revolution in low power long-range wireless communication technologies, including communication protocols like Sigfox and LoRa. Although these low-powered long-range communication protocols provide a reliable solution for low energy consumption and long communication range of end device requirements, The lifecycle of an ever-changing software proves challenging in such edge computing applications. Limitations with communication like low data rate and cycle duty quickly become obstacles to deploying software updates to the end devices. These deployment limitations drive the design of such a system to an approach of utilizing central Cloud computing instead of an onboard processor, even for simple, undemanding operations. Cloud computing, although a great option, comes with its compromises in latency and higher communication dependency as the data must be transferred more often. Although LoRa has a low data rate, it is essential to deploy software remotely, reliably, and continuously into the nodes, especially to think that the end nodes are theoretically deployed in a remote area and can have a long lifespan.

1.1 Related Work

As the LoRa and Firmware deployment over the air has gained popularity in both the research and business fields, a few related works have directly or indirectly inspired or helped with the work done in this thesis. This section points out some of these related works.

1. **Firmware Update Using Multiple Gateways in LoRaWAN Networks** [10]
   In this article, the author has conducted extensive experiments and simulations to investigate the effect of using multiple gateways while doing Firmware updates over the air. The article states a few benefits of using a higher number of Lora Gateways, including higher update efficiency, reduced network energy and a faster update time.
2. **How to make Firmware Updates over LoRaWAN Possible** [11].
   The author review and evaluate the impact of LoRa Alliance's proposed firmware Over the Air process specifications. For this work, a FUOTA simulation tool called FUOTASim was developed to simulate the update process with different FUOTA parameters and conclude the most impactful ones. This work concluded that the suggested LoRa Alliance FUOTA process's phase of "gathering the identifiers of all end devices that are targeted by the update"[12] is unscalable and a potential bottleneck with a higher number of end devices. The work also concluded that using Lora data rate DR5 achieves a 30% reduction in update time and device energy consumption compared to DR0. Also, multicast with class B achieves 17% higher update time and 550 times less energy than class C devices consumption than DR0.
3. **Energy Consumption and Scalability of Transmitting Firmware Updates Over LoRa** [13]
   The thesis introduces a power consumption model based on actual hardware measurements. In this thesis project, many experiments were performed to answer the questions "What is a reliable model to estimate the energy consumption of a firmware update?" and "What is the increase in packet losses during a firmware update?"
4. **Continuous Delivery of Customized SaaS Edge Applications in Highly Distributed IoT Systems** [14]
   This work presents an architecture deployment for a precision agriculture solution that utilizes edge computing and a LoRaWAN network. Future work suggestions included extending the proposed architecture to update the LoRaWAN sensors OTA.

5. **DevOps benefits: A systematic literature review. Software, practice & experience** [15]
   The author reviews the DevOps benefits reported in the literature and maps these benefits with DevOps implementation case studies. The author states that the most reported benefits were "Cross-team collaboration and communication" and "Faster time to market." which aligns with the premises of DevOps of better communication and faster software delivery.

1.2 Thesis Contribution

This thesis's main contribution is to design and implement an optimized DevOps workflow for LoRa applications where computation happens at the Edge. Specifically, the contributions of this thesis are the following:

1. The introduction of a generic design approach for DevOps operations aimed at LoRa-based edge computing applications.
2. The design and implementation of a specific design for parametrizable edge applications require only upgrading part of the firmware.
3. Extending the usability and automation of LoRaWAN suggested firmware updates over the air FUOTA process as a part of a more inclusive CI/CD process.
4. The demonstration of the benefits of the proposed approach with a face detection application.

Earlier research has focused on either DevOps or Firmware updates over the air FUOTA individually. Software industry research and practices seem to evolve in separation from the revolution of embedded computing and IoT, especially LoRa solutions.
This thesis attempts to draw attention to this gap and hopefully trigger further research involving state-of-the-art technologies in software development and IoT industries.

1.3 Thesis structure

This project aimed to design and implement an automated process where software could be remotely deployed into LoRaWAN nodes and determine whether it is possible to implement an entirely automated DevOps pipeline with Continuous Integration and deployment over LoRaWAN. The aim is not to change the design or purpose of a LoRa-based system but to discover the possibility of remotely deploying software to the LoRaWAN end-device when needed.

Chapter 2 goes through essential DevOps concepts, metrics, importance, usages, practices, tools, and research related to DevOps in the software industry.

Chapter 3 LoRa technology background on communication, LoRaWAN network architecture and components like end devices, network servers, gateways, LoRa Alliance guidelines and recommendations, and firmware updates over the air FUOTA.

Chapter 4 introduces the design and implementation of this thesis project design, the DevOps CI/CD implementation, software, image recognition application, remote deployment, the LoRaWAN end device, network server, gateway, and the deployment mechanism.

In the Results Chapter, the outcomes of the implemented system are documented.

Finally, the conclusions of this work and suggestions for future works are in Chapter 5.

2 DevOps

2.1 What is DevOps

The traditional software development process has contributed to widening the gap of organizational isolation between software development (Dev) and production and operations (Ops). DevOps is a new movement in the software industry that introduces a set of principles and practices derived from Agile software development principles. These practices aim to eliminate the gap between the Dev and Ops teams to enable faster software cycles and help solve issues with traditional software development processes. [8][9][16][17]

- The development team (Dev) takes care of the feature's development, bug fixes and software maintenance. This team had no direct communication with customers or shareholders; instead, it depended on the Production and Operations team to communicate with the end-user.
- Production and Operation team (Ops) that worked closer to the customers and shareholders to deliver the product and maintain the infrastructure needed to ensure the stability and functionality of the product (software/system).

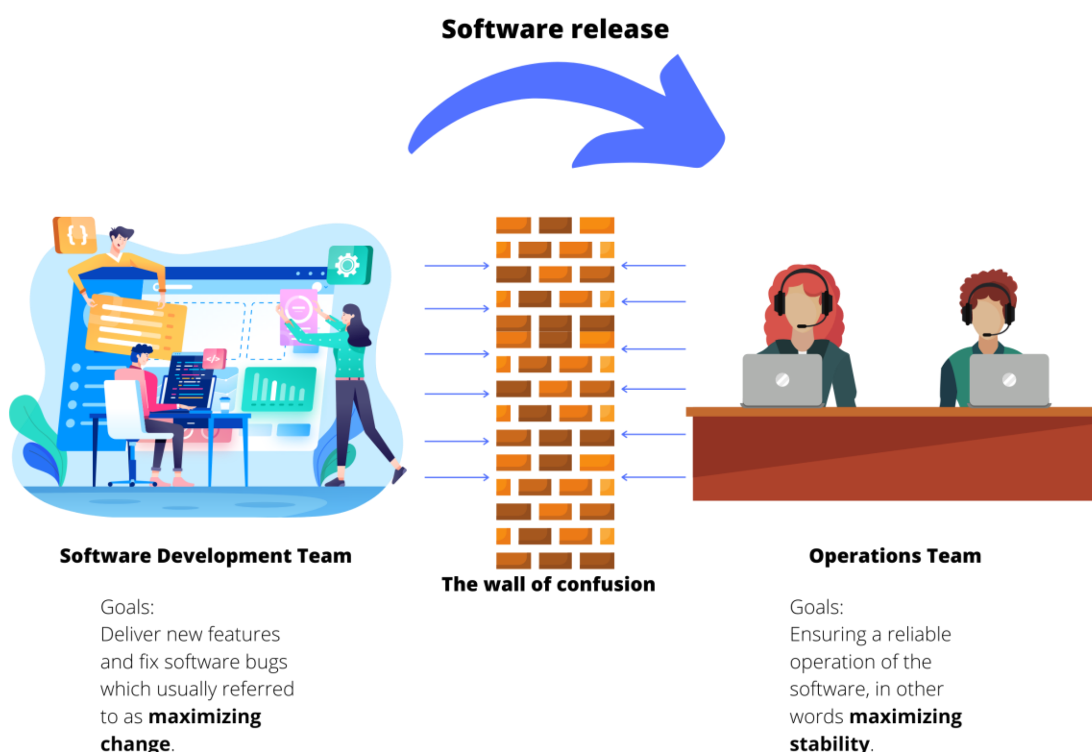

Figure 2-1 Organizational and process introduced Wall of Confusion between Dev and Ops teams

Those two teams traditionally worked in silos which hindered their ability to move fast and limited the visibility and communication across the software development process. The operations team often has minimal knowledge about the software development before the actual software release, which tremendously affects their primary goal of stability. Reduced stability in the software in question meant questionable reliability and more challenging incident recovery.

This traditional software development model has some issues, including:

- Software development cycles were exceptionally long, which meant that issues and new features took a long time to get addressed due to the inefficiency of the process and the lack of communication and visibility. So, bugs, for example, are not discovered until the software is released and create issues in the production environment.
- Due to the manual and rigid software release/deployment process, there are ongoing issues with integrating and deploying new versions. This process also contributes to extending the development cycles. Handovers to the next team also meant that time was wasted waiting for other teams to hand over, which translates into a noticeable waste along the process.
- The organizational/functional segregation of teams, their goals and their responsibility created a conflict of interest between them.
- This segregation and lack of communication also created ongoing issues with reliability; for example, when the development team tests the software functionality in an environment (Server) with specific settings and everything seems to work as it should, but after handing over the release to the operations team which is using a different environment in their servers unexpected issues are possibly rising.

These issues result in lower quality and reliability of the software and a slower and inefficient development process.

2.2 DevOps Metrics

Measuring the success of DevOps with an organization can be achieved by monitoring metrics and KPIs that show software deployment and quality performance. The importance of each metric depends on the teams working with the software pipeline, the organization, and its goals. These metrics can show how well DevOps is adopted within an organization and how the organization can improve. [17][18]

The DevOps metrics have been classified differently in the literature. The author of [17] suggested 3 overlapping metrics categories with metrics affecting one of 3 scenarios. Velocity + Stability, Quality + Stability or Velocity + Quality + Stability.

1. **Velocity Metrics:**
   These include metrics like Deployment duration, Deployment frequency, Change volume, Lead time, Cycle time, Deployment failure rate and Environment provisioning time

2. **Stability Metrics:**
   Mean Time to Recovery (MTTR), Deployment downtime, Change failure rate, Incidents per deployment, Unapproved changes, Number of hotfixes and Platform availability.

3. **Quality Metrics:**
   Defect density, Test automation coverage, Defect ageing, Code quality, Unit test coverage, Code vulnerabilities, Standards violations, and Defect reintroduction rate.

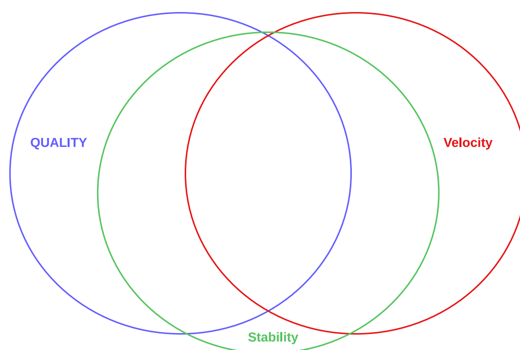

Figure 2-2 Relationship between Velocity, Quality and Stability [17]

The author in [18] suggests different dimensions of DevOps metrics, including dimensions affecting staff, business, customers, process, and technology.

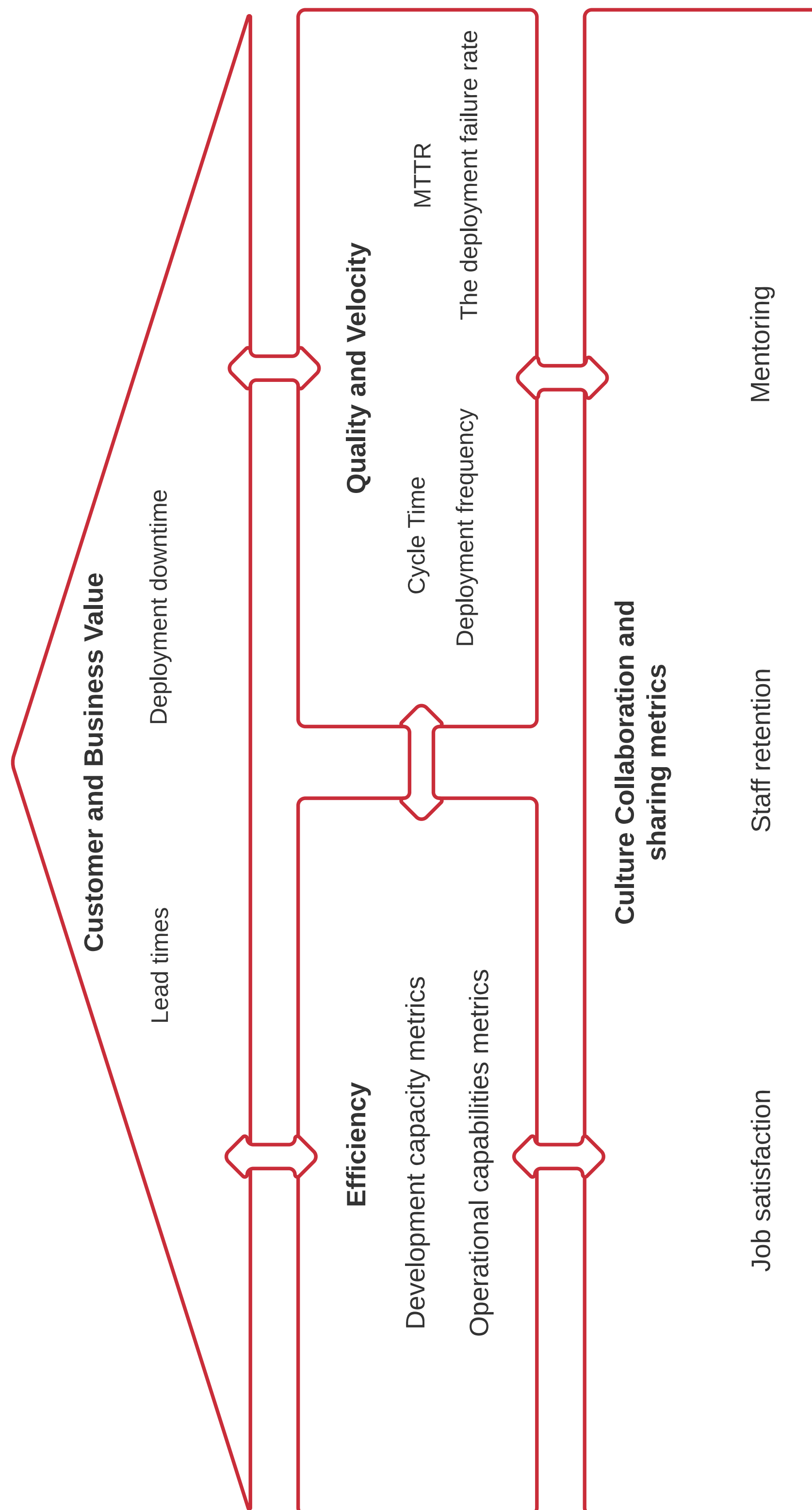

Figure 2-3 DevOps metrics value Dimensions suggested [18]

2.3 DevOps process

Unlike the traditional software development model, the DevOps lifecycle is a continuous process where all teams are involved in the entire process. These process phases can be modified to best suit the organization's workflow and needs. The focus is on the process's stability, velocity, and software quality. [17]

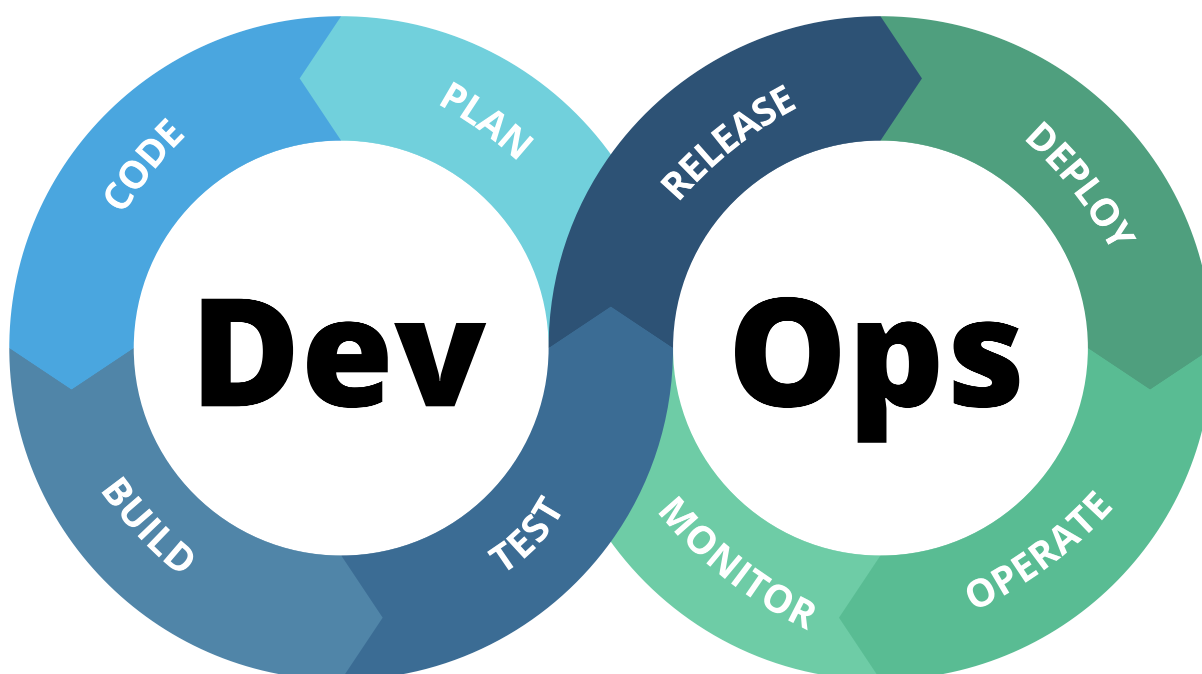

Figure 2-4 DevOps Process

DevOps as a process can be achieved by adopting the following practices or concepts in the process of software development [17]:

- **Source Code Management (SCM)**
  Source Code Management enables multiple developers to work with the code concurrently regardless of geographic location or time. SCM system documents all the changes done to source code, and it is crucial to track features and bugs with the software later. This detailed tracking is essential for automating the process and fast recovery in case of failure, as the issue can be easily pinpointed and rolled back if needed. The SCM system would also allow maintaining different branches for bug fixes, new features, and minor and major releases.
  Integrating SCM systems with DevOps processes enables integration and automation robustness.

- **Code reviews**
  The code reviewing process ensures the improvement of software quality. Reviews are done by other software developers or automated static code quality tools, which helps to identify common bugs and issues with the code quality before merging and integrating it into the mainstream.

- **Configuration Management**
  Configuration Management is done to identify, manage, verify, and maintain any configuration for the software and hardware involved throughout the process. This practice achieves a status where all teams clearly understand configurations used throughout the process, including other teams' configurations.

- **Build management**
  Build management is where the source code and all required dependencies, including other software and hardware, are put together and built into functioning software. Building the software with every change in the source code is essential to pinpoint issues effectively.

- **Artefacts' repository management**
  This management system is dedicated to keeping and managing the binaries created at the build management system. Saving those artefacts is needed to ensure builds are repeatable, keeping those for later testing and investigating potential hidden issues.

- **Release management**
  Release management facilitates and manages the software's development, testing, deployment, and maintenance. The automation of this process relies on the automation of all those other processes.

- **Test automation**
  Manual testing is very tedious, expensive, and time-consuming. Test automation automates all scenarios with all new code pushed to source control. This automation means it is possible to extensively and reliably test every new piece of code and pinpoint issues discovered during testing. Overall, software quality can be substantially improved by test automation.

- **Continuous Integration**
  When working with different pieces of software or multiple software/dependencies, Continuous Integration is needed to ensure the quality and functionality of the code as early as possible.

- **Continuous Delivery**
  After successful integration, the product is automatically delivered to a staging environment where issues with business logic can be discovered during the user acceptance tests.

- **Continuous Deployment**
  Deploying to production is the last point of source code changes after building, testing, integrating, and delivering. Doing this requires the automation of the entire process.

- **Routine automation**
  Automate any routine and repetitive tasks. Tasks like updating version numbers, creating a new branch for a release, and copying user-interface localization to or from a database can all be automated to save time and increase the quality of the results.

- **Infrastructure as Code (IaC)**
  IaC is an integral part of configuration management. IaC aims to identify the entire infrastructure used in testing, building and deployment and saves it as a file used by the configuration management system to replicate an infrastructure setup such as production environment staging (virtual servers, clients, and networks) that is needed in the process.

- **Key application performance monitoring/indicators**
  Performance metrics are essential to monitoring the software's quality and the development lifecycle. Metrics like uptime, downtime, mean time to recovery, mean time to detect errors, deployment frequency and lead time are needed when evaluating and recognizing specific trends and issues with the overall process and helping to improve the development pipeline continuously.

These practices can be associated with on or phases of the process. Performance Monitoring and Routine Automation are required throughout the entire process, as shown in *Figure 2-5*

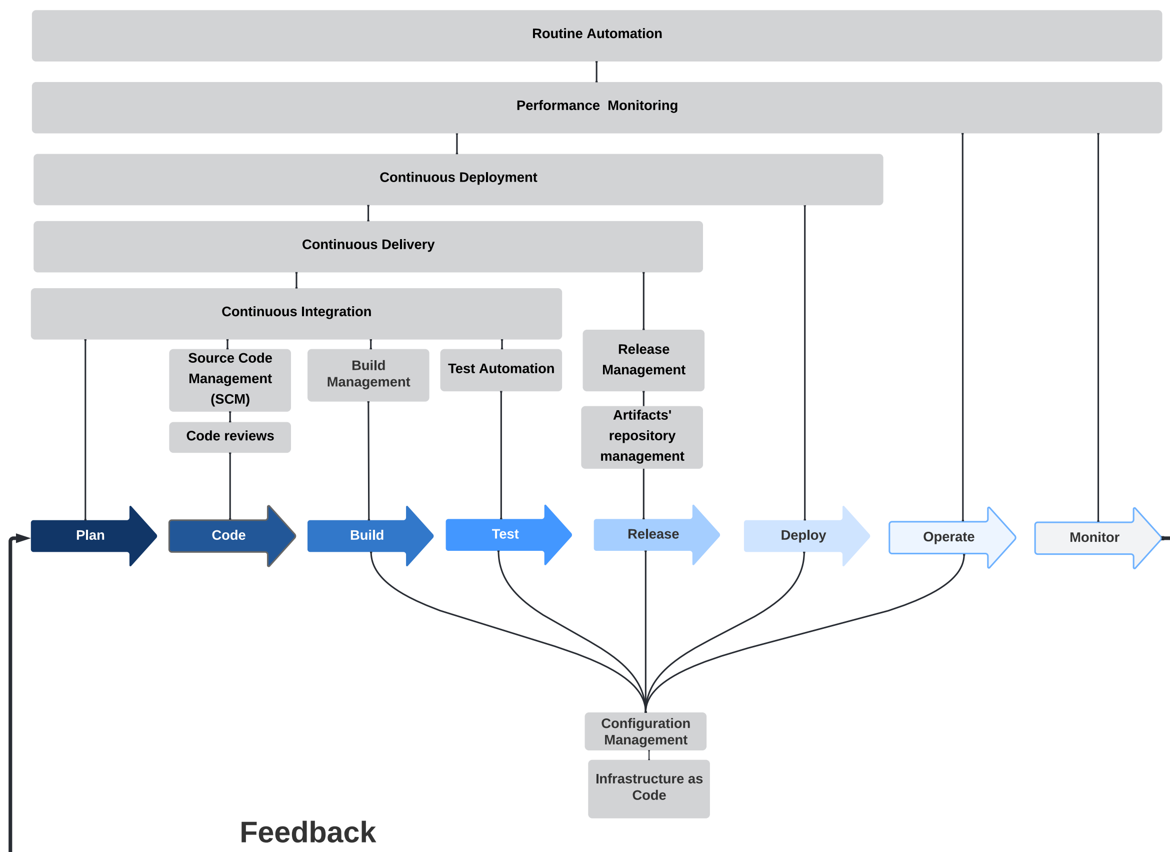

Figure 2-5 DevOps Process Phases Association with Practices

Adopting these practices requires cultural adaptations throughout the organization and using supporting tools and infrastructure. Besides the cultural adoption of DevOps, it requires specific tools to practice DevOps successfully. For instance, to be able to practice Test and Build automation, an automation server is needed. GitHub, Bitbucket, or any other version control tool would be required to practice Source Code Management.[17]

2.4 DevOps Adoption Benefits

The DevOps movement aims to break down communication and collaboration barriers between teams involved in the software development process. This goal refed to as "Tear Down the Wall", which is achieved by Adopting a culture in the organization where everyone is collaborative. The responsibility in the DevOps process is imposed from End-to-End. So, teams would need to work together to achieve a common goal instead of conflicting teams' goals. The End-to-End responsibility also means Continues improvement is encouraged as everyone needs to adapt to changing circumstances and modern technologies. Automation implementation whenever possible means a shorter and more stable development cycle, so instead of weeks or months with the traditional development process, the development cycle is measured in days, hours or even minutes. The DevOps process focuses on the customer's needs and continuously pivots when needed. The process takes a fail-fast strategy which helps discover issues as soon as possible, learn from them and fix them quickly. DevOps teams are involved at every stage of the software development cycle. The team is cross-functional, and responsibilities are shared across teams.[15][18]

The Accelerate State of DevOps Report by the DevOps Research and Assessment (DORA) [18] team at Google Cloud has been researching and publishing the effects of following DevOps practices and adaptation on organizations' performance. The report categorizes organizations into four categories based on the answers provided by IT professionals on software delivery performance metrics, as shown in Table 2-1.

DORA's previous yearly reports consistently improved performance and reliability results for organizations with higher software delivery metrics than those with lower results. The Accelerate State of DevOps 2021 shows that organizations within the Elite performers made 973 times more code deployments and had 6570 times faster lead time from committing code to deploy than the low performers' group. The report also shows that the Elite group has three times less failure rate and 6570 times faster recovery time from failures than the low performers.[18]

Table 2-1 Software Delivery performance results per performance category [18]

Software Delivery Performance metrics	Elite	High	Medium	Low
Deployment frequency	Multiple deploys per day	Between once per week and once per month	Between once per month and once every 6 months	Fewer than once per six months
Lead time for changes	Less than one hour	Between one day and one week	Between one month and six months	More than six months
Time to restore service	Less than one hour	Less than one day	Between one day and one week	More than six months
Change failure	0%-15%	16%-30%	16%-30%	16%-30%

3 LoRa

3.1 What is LoRa?

Long Range (LoRa) is a Radio Physical Layer (PHY) of a Low Power and Wide Area technology (LPWA) communication that uses chirp spread spectrum (CSS) modulation to represent the payload bits in multiple chirps. [22]

Chirp is a signal with continuously various frequencies. The rate of those chirps is referred to as the spreading factor (SF). The spreading factor impacts the communication performance as a higher spreading factor means lower chirp rate and, therefore, lower data transmission rate and more extended transmission range. The opposite is correct for the lower spreading factor. LoRa uses SF between 7 and 12. [22]

LoRaWAN is the Media Access Control layer (MAC) defining and maintaining the LoRa protocol specifications. These specifications are defined and maintained by the non-profit association LoRa Alliance. [19]

Despite the low data rate, the long-range and low-power nature of LoRa modulation gives an advantage in a wide range of Internet of things applications that require end devices with high energy efficiency and long-range communication. Such applications can utilize LoRa in peer-to-peer connections or within a LoRaWAN network. LoRa has proven efficient in smart cities, smart agriculture, electric metering, gas metering, smart homes, and even cleaning services.

Table 3-1 LoRa DataRate parameters [23]

LoRa DataRate (DR)	Configuration (SF/bandwidth)	Indicative physical bit rate (bit/s)	Max payload size (bytes)
0	SF12 / 125 kHz	250	51
1	SF11 / 125 kHz	440	51
2	SF10 / 125 kHz	980	51
3	SF9 / 125 kHz	1760	115
4	SF8 / 125 kHz	3125	222
5	SF7 / 125 kHz	5470	222
6	SF7 / 250 kHz	11000	222

3.2 LoRaWAN Architecture and Components

LoRaWAN enables LoRa devices to connect to internet applications over long-range wireless communication. In the next section, an introduction to the components of the LoRaWAN network.[19]

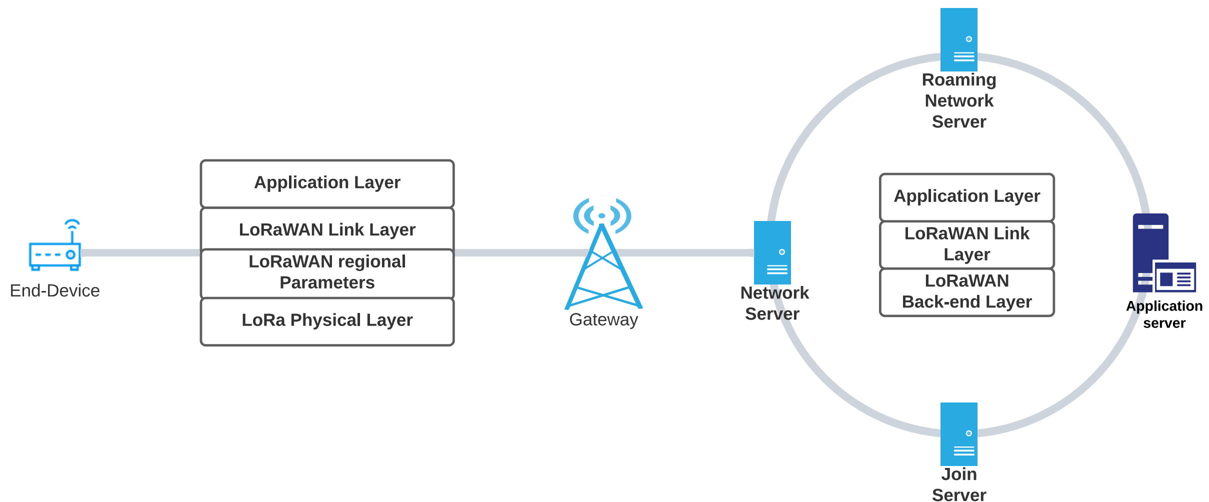

Figure 3-1 LoRaWAN Network Architecture

- The network Server is the core of every LoRaWAN Network. The server manages the LoRaWAN network, including gateways, end-devices, message routing and integrations, adaptive Data rate control and acknowledgement of messages. The network server is responsible for message deduplication as well.
  There are various providers of LoRaWAN servers with public, on-premises, and cloud hosting options.

- Join Server (LoRaWAN v1.0.4+) is required to processes join-request messages from end nodes and manage root and session keys.

- LoRa Gateway's primary function is to connect end devices and network servers, and it needs to be connected to the Internet to communicate with the network server.

- Application Server The application processes data from end devices and can schedule downlink messages sent to the end devices when needed.

- End Device or LoRa node is the Edge of the LoRaWAN where either monitoring or actuating is required.

3.3 LoRaWAN End Device Classes

The end device or node is usually a sensor, or an actuator deployed within the LoRa RF range from the nearest gateway. LoRaWAN specification defines 3 different classes of end devices Class A, Class B and Class C depending on the application.[19]

- **Class A** end devices have two short receive slots following each uplink transmission. Class A is primarily suitable for applications where the downlink messages can wait for the following uplink message, so the device opens the short downlink listening slots right after transmitting its uplink. Class A is the most energy-efficient class.

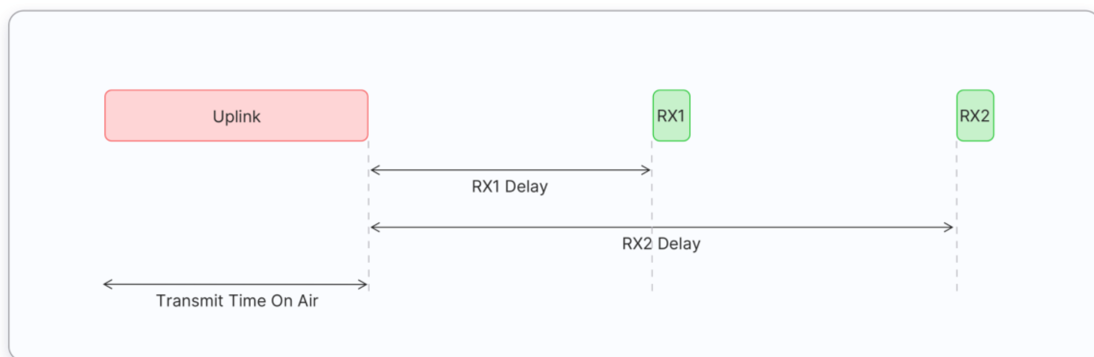

Figure 3-2 Class A receive windows as appears in [21]

- **Class B** devices have scheduled receive slots. The difference with Class A that besides the two receive slots, the downlink slots can be scheduled. This scheduling requires a time-synchronized downlink (Beacon frame) to be received from the gateway to confirm that the device is listening.

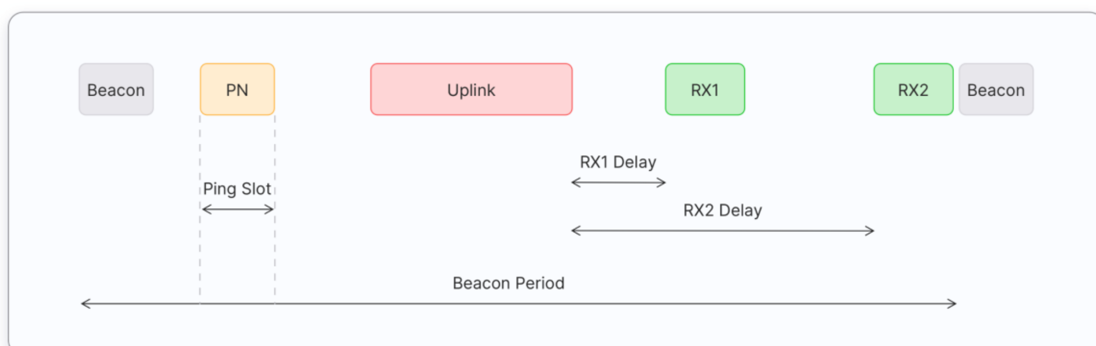

Figure 3-3 Class B receive windows as appears in [21]

- **Class C** end devices have maximum possible receive slots. A Class C device should allow continuous open receiving slots. Class C devices have the lowest latency for downlink messages as the device is continuously listening to downlinks. The device's always ready for downlink state comes with the cost of higher energy consumption compared to Class A and Class B.

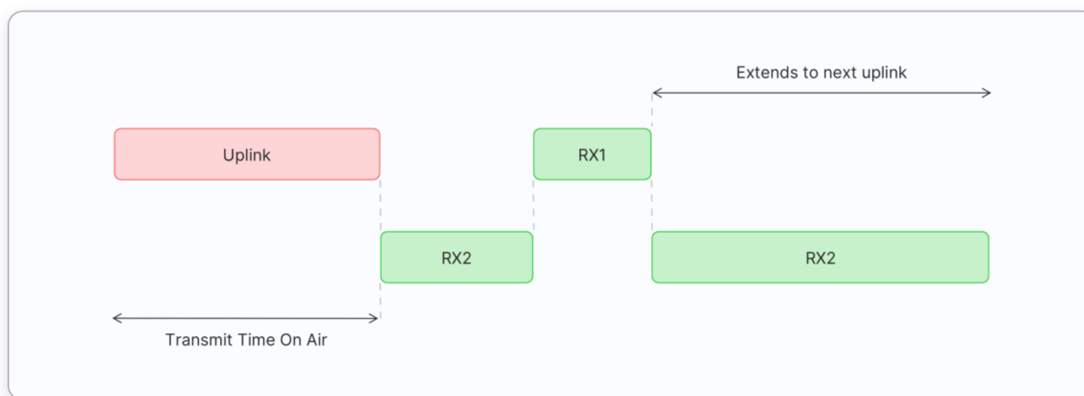

Figure 3-4 Class C receive windows as appaears in [21]

3.4 End Device Activation in LoRaWAN

The end device needs to be personalized and activated for the end device to join the LoRaWAN network [19]. After activation, the end device saves the following information:

- **Device Address (DevAddr):** A 32bits identifier of the end device within the current LoRaWAN network. To be compliant with LoRaWAN (DevAddr) identifier must include (AddrPrefix), which should be derived from the Network Server unique identifier (NetID) that LoRa Alliance has allocated to the server. This restriction does not apply in the case of private network servers.

- **Network Session Key (NwkSKey):** This device-specific key is used by the Network Server and the End Device to calculate and verify message integrity code to ensure data integrity. This key is used to encrypt Mac-specific data frames sent on Port 0.

- **Application Session Key (AppSKey):** This device-specific key is used by the Application server and end device to encrypt the application-specific data frames.

The LoRaWAN specifications [19] define an activation process needed for activating end devices. The activation process depends on the activation method used. Either of these methods can be used for activation:

- **Over-The-Air Activation (OTAA)**
  The end device shall have a globally unique end-device identifier (DevEUI), the Join Server identifier, and an AES-128 key (AppKey) To use the OTAA method. In OTAA, the end device initiates the join procedure by sending a Join-Request frame that contains JoinEUI, DevEUI and DevNonce. DevNonce is a counter starting at 0 and increments with the Join requests the device has sent. Join server saves the DevNonce for each device, so it ignores the Join-Request frame of the device if DevNonce is not incremented. If the end device was permitted to join the network, the Join-server sends a Join-Accept frame which allows the end device to calculate the Network Session Key (NwSKey) and the Application Session Key (AppSKey).
  In the case of Class C devices, the end device must send a confirmed uplink frame after receiving the Joint-Accept frame to finalize the join procedure.

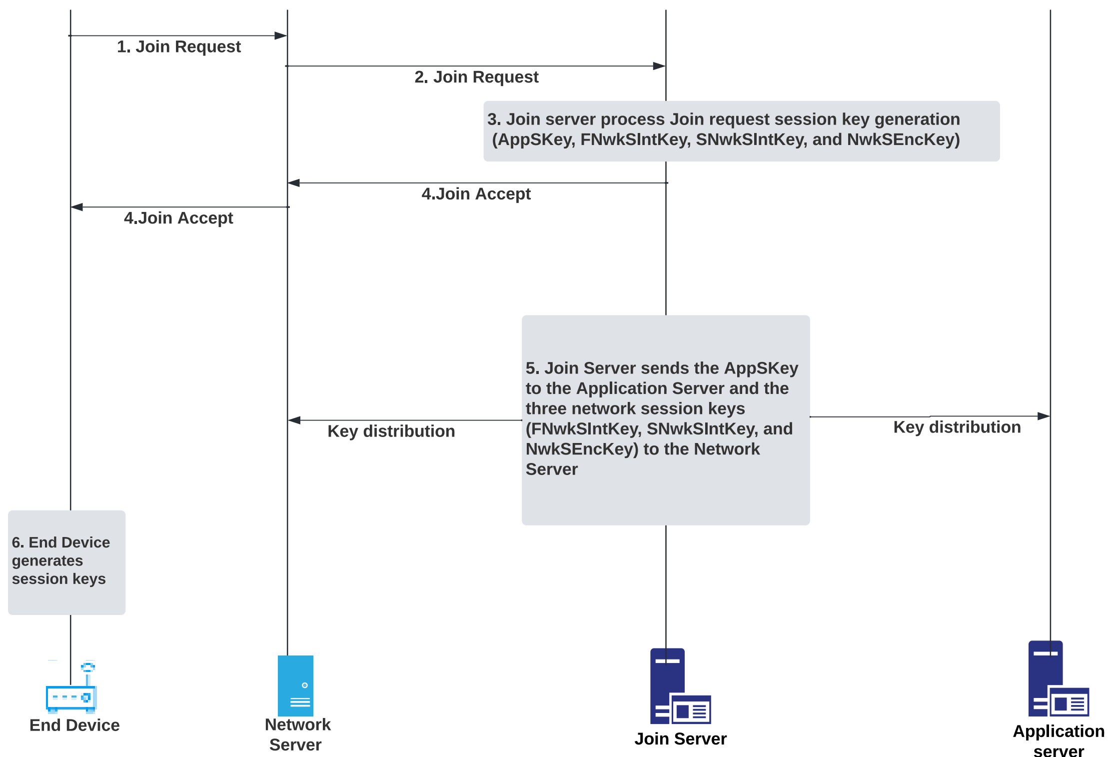

Figure 3-5 OTAA activation in LoRaWAN v1.1

- **Activation by personalization (ABP)**
  This activation method requires the device to have the following pre-allocated identifier and keys: Device Address (DevAddr), Forwarding Network Session Integrity Key (FNwkSIntKey), Serving Network Session Integrity Key (SNwkSIntKey), Network Session Encryption Key (NwkSEncKey) and an Application Session Key (AppSKey). This method does not require sending Join-Request or Join-Accept frames, and the device can join the network as soon as it is powered up.

Using OTAA is more flexible as the required keys to join the network are created dynamically and not predefined and saved into the end device like in ABP activation. An OTAA end device can join another LoRaWAN network without needing to reprogram the device.

3.5 End-Device Firmware Update Over The Air (FUOTA)

Updating the end device firmware has become a requirement in IoT to push firmware batches and security updates whenever needed. End-device update over the air procedure is highly dependent on the architecture of the LoRa application and end-device architecture. This process can be implemented as a part of the application layer running on top of a LoRaWAN network. LoRa Alliance FUOTA recommendations [12] define the outlines of the network architectures and the FUOTA process.

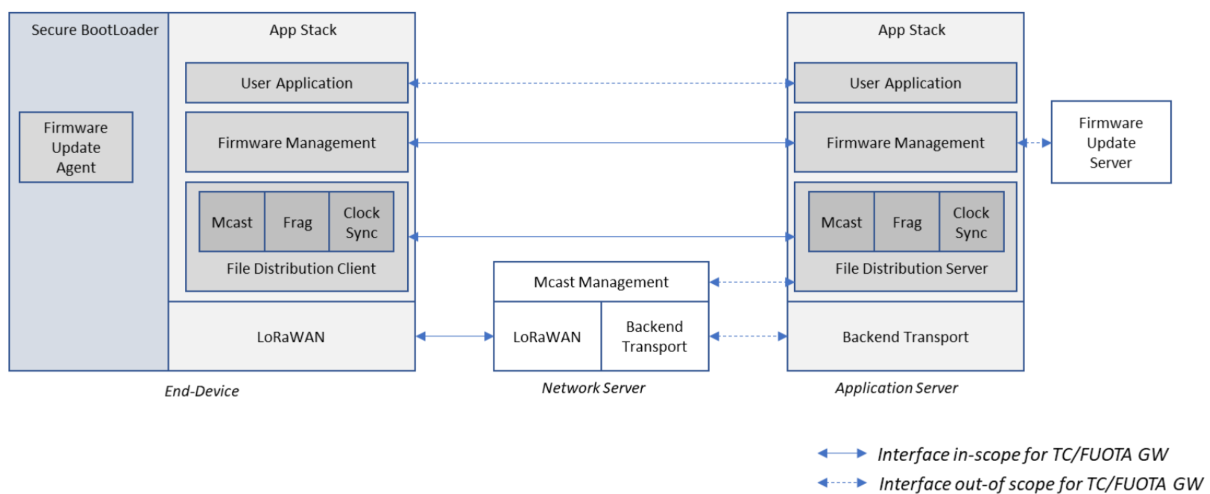

Figure 3-6 LoRaWAN FUOTA network architecture as appears in [12]

Firmware update over the air is complex and can be achieved with variable technologies and methods. High-level recommendations and process descriptions by Semtech [24] and LoRa Alliance [12] can be summarized in *Figure 3-7*. Besides documents published by LoRa Alliance and Semtech, the FUOTA process has not yet been standardized. FUOTA steps are device-specific, and implementation can vary dramatically according to end-device MCU architecture, memory capacity, operating system, bootloader, whether the firmware is modular or not, and whether the device is utilizing a cryptographic hardware accelerator or not. This lack of standardization leaves the actual implantation to the system and end-device designers to decide.

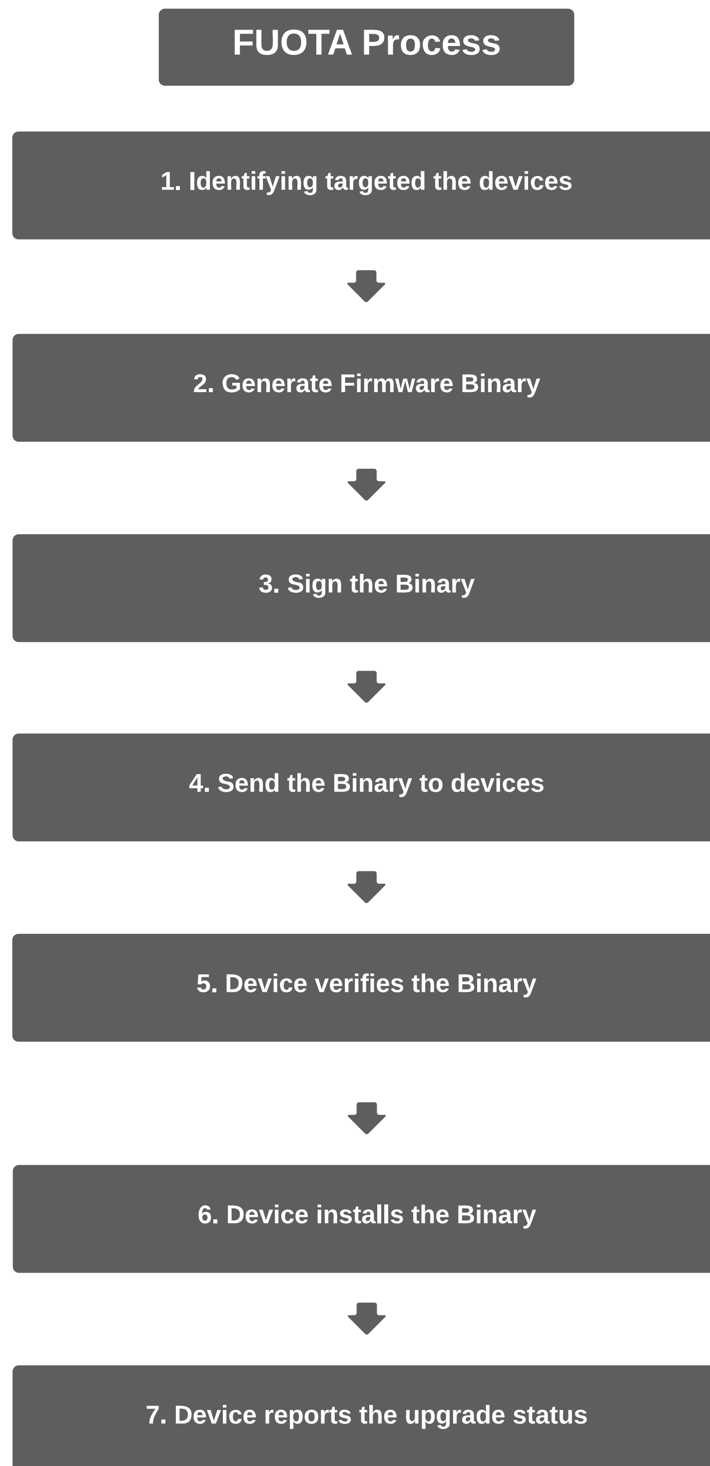

Figure 3-7 FUOTA process as suggested in [24]

3.6 LoRa Multicast

The lack of FUOTA process standardization and the low bit rate of LoRa does add some challenges to the FUOTA process. However, the low bit rate issue is solved using LoRaWAN radio multicast, where multiple devices can receive a packet transmitted by the network. Radio multicast allows the packet to be sent only once and received by all the targeted end devices simultaneously.[20]

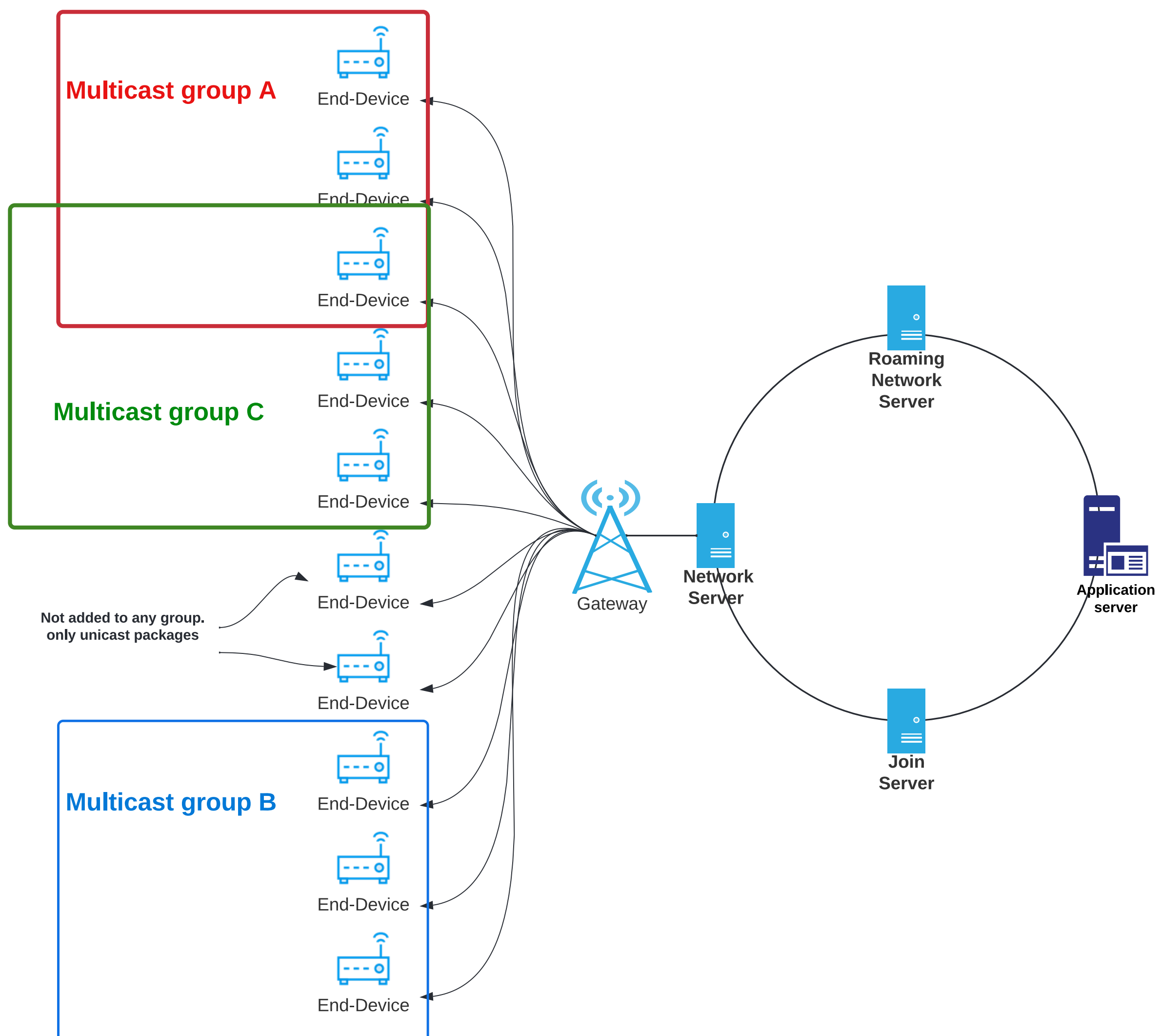

Figure 3-8 LoRaWAN Multicast

Multicast can function with end-devices of Class B and C only. Sending each firmware update downlink once as a multicast message, so 1000 end-devices receive it at once, would be more efficient than sending the same downlink 1000 times as unicast. Class A devices are not suitable for multicast and, therefore, not suitable for FUOTA for the following reasons:

- Class A devices do not receive downlink messages unless there is an uplink message, so for sending firmware in 100 downlink messages, the end device would need to send a 100 uplink which significantly raises the power consumption if uplinks are sent only to check for downlinks.
- Class A uplink requirement also significantly slows down the firmware receiving process as each downlink message needs to wait for the next time the end device sends an uplink message.
- Redundantly transmitting uplink messages only to receive the downlinks contributes to hitting the end device's duty cycle limits and imposes an overhead on network traffic.

Multicast requires end devices to be set up as a part of the multicast group. This setup can be remotely achieved in LoRaWAN. LoRa Alliance has defined the process of LoRaWAN Remote Multicast Setup with a "Multicast Control" package which is used to:

- Remotely set or remove multicast security keys to the end device.
- Report the multicast groups to which the device has been added.
- Program a Class B or Class C multicast session.

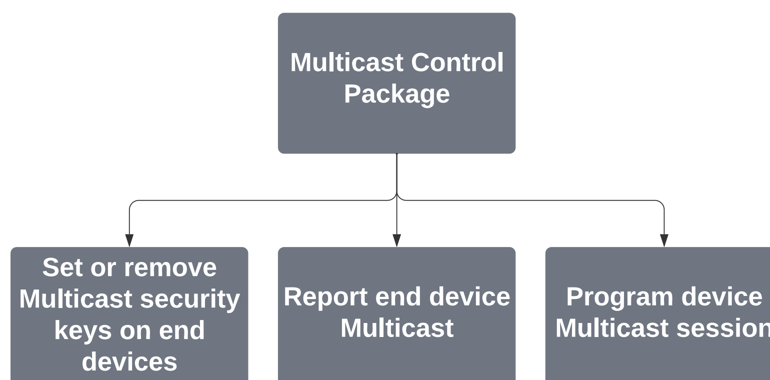

Figure 3-9 Multicast Control Package functions

After the Multicast Setup, the end device saves the McGroupID, the Multicast address, the multicast group key, and the Frame counter.

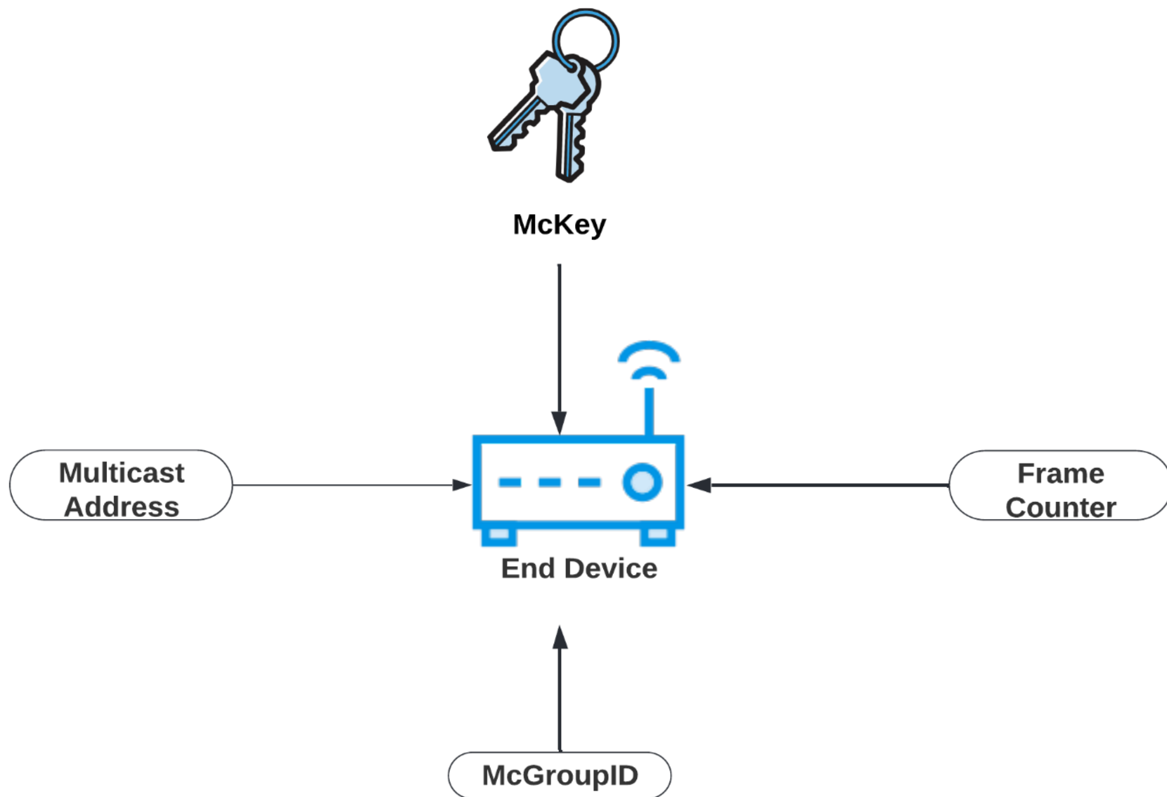

Figure 3-10 End device parameters set by multicast setup

- McGroupID: an end-device-specific integer of the index of the multicast group simultaneously supported by the device. The value of this integer can be 0:N-1, where N is the maximum number of multicast groups supported by the device simultaneously.
- Multicast address: end device saves a 4bytes network address of the multicast group. This address is shared for all the devices in the multicast group.
- Multicast Group Key (McKey): a Multicast-group-specific key which is used to derive the Multicast Application Session Key (McAppSKey) and the Multicast Network Session Key (McNwkSKey).
- Frame counter to register the count of frames sent by the multicast group.

3.7 Challenges of remote Firmware deployment over LoRaWAN networks

Given that the LoRaWAN protocol was designed for applications with low data rates and limited data transfer requirements, remote firmware deployment over LoRaWAN becomes challenging in many ways. Packets loss and duty cycles would mean taking hours or even days to transmit a small file. These remote deployments are affected by many parameters, including the following:

- **Duty cycle limits**

  The duty cycle can be defined as a proportion of time during which the network is used. The time of transmitting is defined as Airtime. LoRa radio communication is regulated by European Telecommunications Standards Institute standards in Europe. According to these standards, the Duty Cycle of a Lora device is 1% which means that for every second the device is transmitting, the device should not transmit for the next 99 seconds. This limit is an issue when trying to transmit frequently, as in the case of FUOTA.

- **Deployed file size**
  Typical firmware can range anywhere from a few KB to tens of MB, if not more. A larger firmware causes to reach Duty Cycle limits. For example, transmitting a 1 KB firmware over LoRa in DR0(max packet size of 51 bytes) requires sending at least 20 downlink messages, while the duty cycle limits the number of packets with such airtime to 12 packets/hour. This rate means the firmware of 1KB needs around 1.6 hours to be sent [25]. A 10 MB Firmware with the same settings above would need at least 192308 downlinks and more than 16000 hours.

- **Data rate used**

  A higher DR allows larger packets (up to 222 bytes/packet), but it also has a much lower range. Implementing adaptive data rate solutions can help adjust the DR according to the distance.

- **The number of updated end devices to updated**

  The higher the number of devices to be updated, the longer it would take to finish the update (Duty Cycle). This limitation can be elevated by using multicast groups and Multiple gateways.[10]

Implementing Continuous Deployment to LoRa end devices can be challenged by the low data rate, duty cycle and package loss when sending updates over LoRaWAN.

For example, deploying a firmware or a file to an end device with a size of 10 KB while using DR0 mode with a maximum payload of 51bytes would require fragmenting the file into 197 fragments and sending at least 197 downlinks to each end device. This 10Kb file deployment becomes even more challenging when deploying to tens, hundreds, or thousands of end devices. There needs to be a mechanism to recover from lost packets while minimizing the redundant resending of the fragments.[25]

4 Case: CI/CD over LoRa

4.1 Design

The main goal of this project is to implement a DevOps Continuous Integration and remote deployment pipeline for a LoRaWAN node. This project focuses on LoRaWAN applications requiring end devices to do data processing. The case used for this project was to update a face recognition application deployed to Raspberry Pi end-devices used for access control. The access control is done by a face recognition application that depends on a pre-set of reference images of the persons with access rights. The image capturing and processing are done on the Raspberry Pi node. The Raspberry PI controls a door locking actuator based on the saved authorized person reference images encoding. The goal is to find the possibility of updating the face recognition application and its reference images remotely so that the nodes can allow access for authorized people as the reference images are changing. The application and all new faces encodings must be tested with test automation and deployed to end devices over a LoRaWAN network. The application has a web interface to allow system administrators to manage the access authorization by adding or removing personnel to the designated access area.

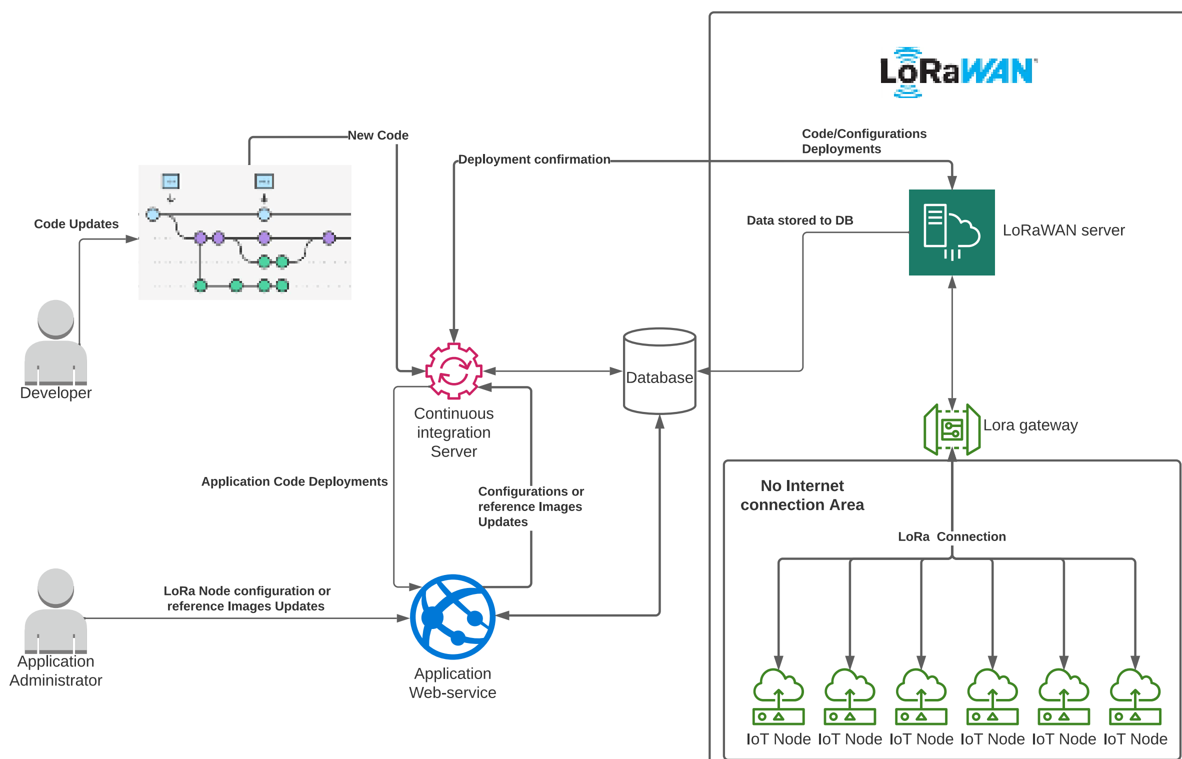

Figure 4-1 LoRaWAN Edge computing CI/CD Design

4.2 LoRaWAN Setup

The LoRaWAN network in this project consists of a LoRaWAN gateway, the nodes, and the LoRaWAN Network Server. The setup can vary depending on the gateways, end-devices, LoRaWAN server, and the LoRaWAN application, but for this thesis's scope, the setup details are mentioned only for the current project hardware and application. The gateway device used is a Laird RG186 gateway. Each node consists of a Raspberry PI 3 B+ device, a Raspberry Pi Camera rev 1.3 and a LoRa Node PHAT module from Pi Supply. The LoRa Node PHAT Module is based on a RAK811 module, integrating a Semtech LoRa module SX1276 and an stm32L microcontroller. Using AT commands, this integrated stm32L microcontroller in the RAK811 controls the LoRa SX1276 chip through the Raspberry PI UART interface.

4.2.1 LoRaWAN Server and Gateway Setup

As discussed earlier, there are multiple operating options and providers of LoRaWAN server service. The setup process varies significantly depending on the hosting server option. However, the functionality and operating of the LoRaWAN server are similar even across different service providers as the primary usage of adding and managing Gateways, end-devices, HTTP and MQTT integrations functions the same way regardless of the LoRaWAN server provider used. So, a few options were assessed, including a private LoRaWAN server and a public cloud server.

Figure 4-2 loriot.io network server geographical locations

For this project's LoRaWAN server, LORIOT Community Network Server was used. The free network server can be used to build demo applications and LoRaWAN networks. The setup starts by registering an account at https://loriot.io/ after choosing the geographical server location for the project. This project's geographically closest available community network server was in Frankfurt, Germany (EU1). All the network settings can be adjusted from within the web interface of Loriot. The community account has a Sample application and a Sample network by default. The gateway can be added to the network simply by choosing Add Gateway, then selecting the gateway base platform as in the following Figure. In this project, Laird RG1 was the base platform for the gateway. The Gateway eth0 MAC address is needed in this configuration. In this case, the Gateway manufacturer provides it and can be found on the backside of the gateway device. After adding the gateway, the region configurations can be set in the Loriot server.

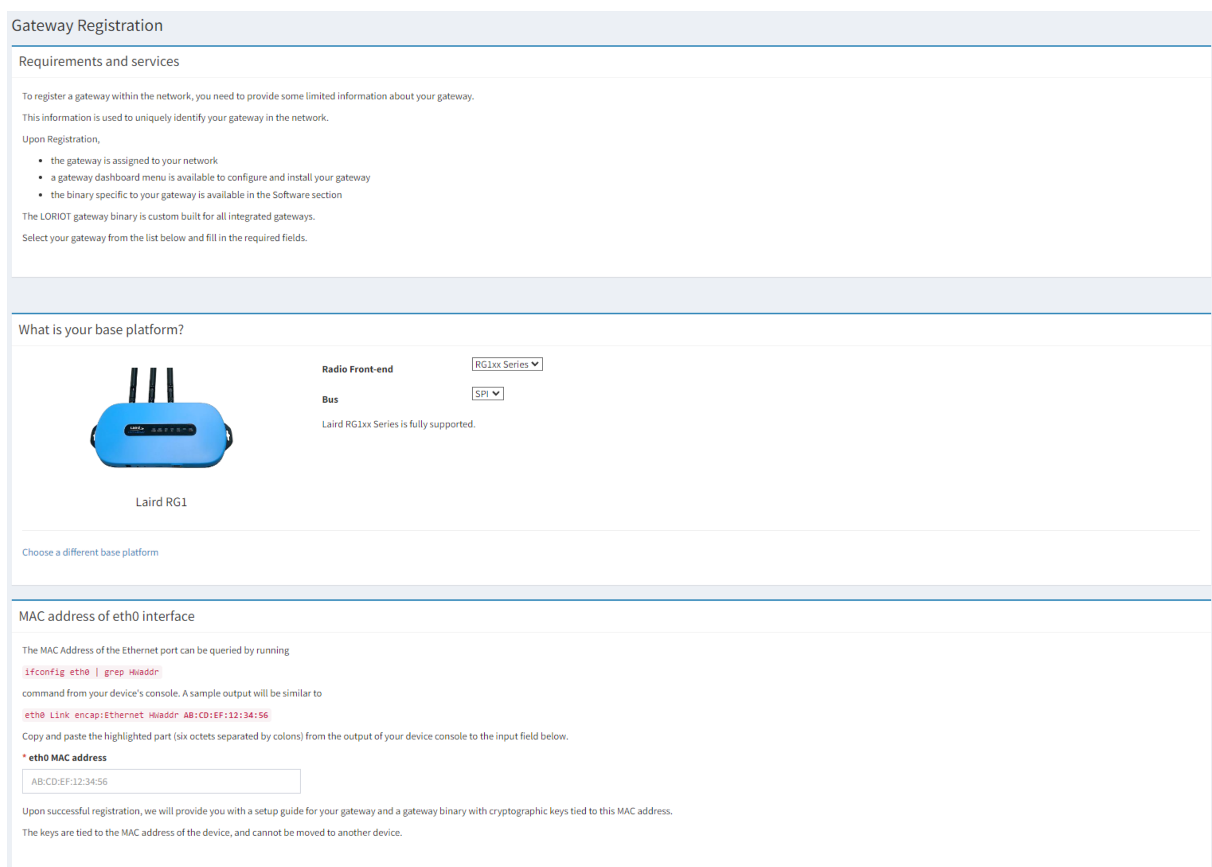

Figure 4-3 Laird RG1 Gateway Registration

The LoRa network server is set to (LORIOT.io EU) on the gateway web interface. After configuring the Loriot server with the correct Gateway parameters, the gateway needs to be set to forward messages to the used network server (Loriot.io). In this case, changing the gateway configuration after connecting it to the local WAN network can be done through the gateway CLI or by accessing the configuration URL (Gateway local IP address) with a web browser and providing the username and password. When the gateway setup is done correctly on both the LoRaWAN Server and the gateway sides, both sides should show the gateway's online and connected status. After setting up the gateway, there needs to be a LoRaWAN application which, in this case, the Loriot community account comes by default with one sample application (SampleApp) and does not allow creating more applications without upgrading the service.

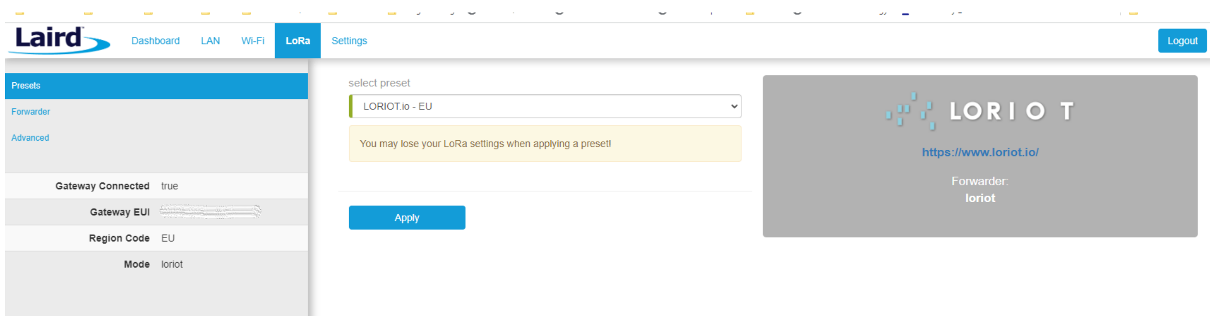

Figure 4-4 Laird Gateway forwarder setting to Loriot.io

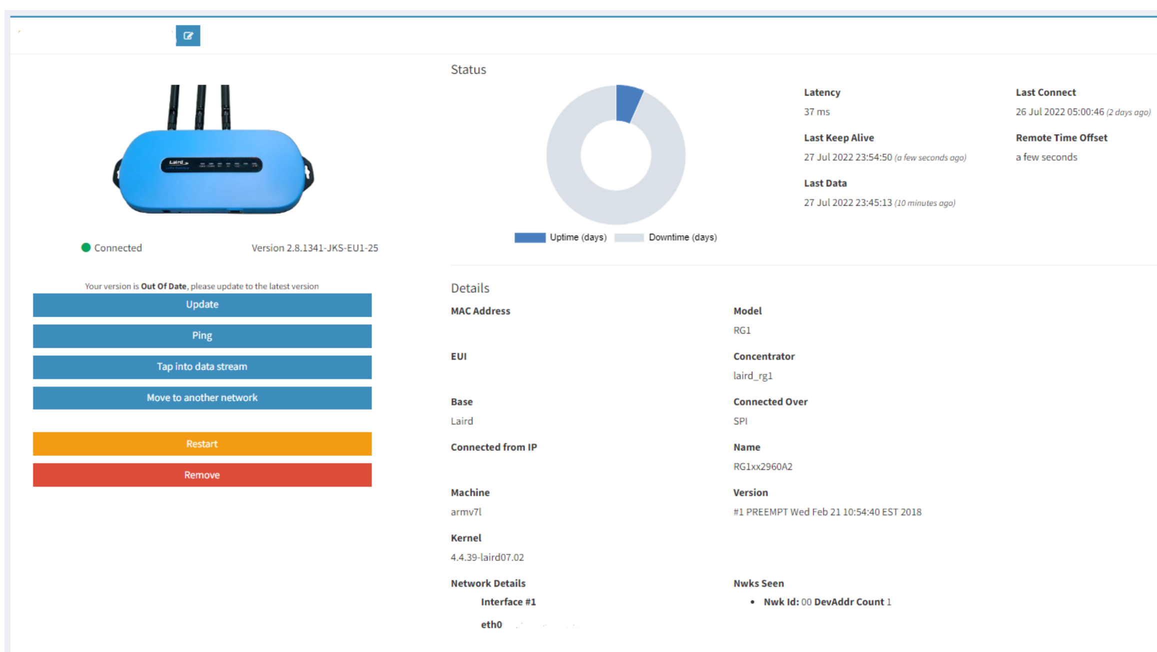

Figure 4-5 Loriot server registered gateway dashboard

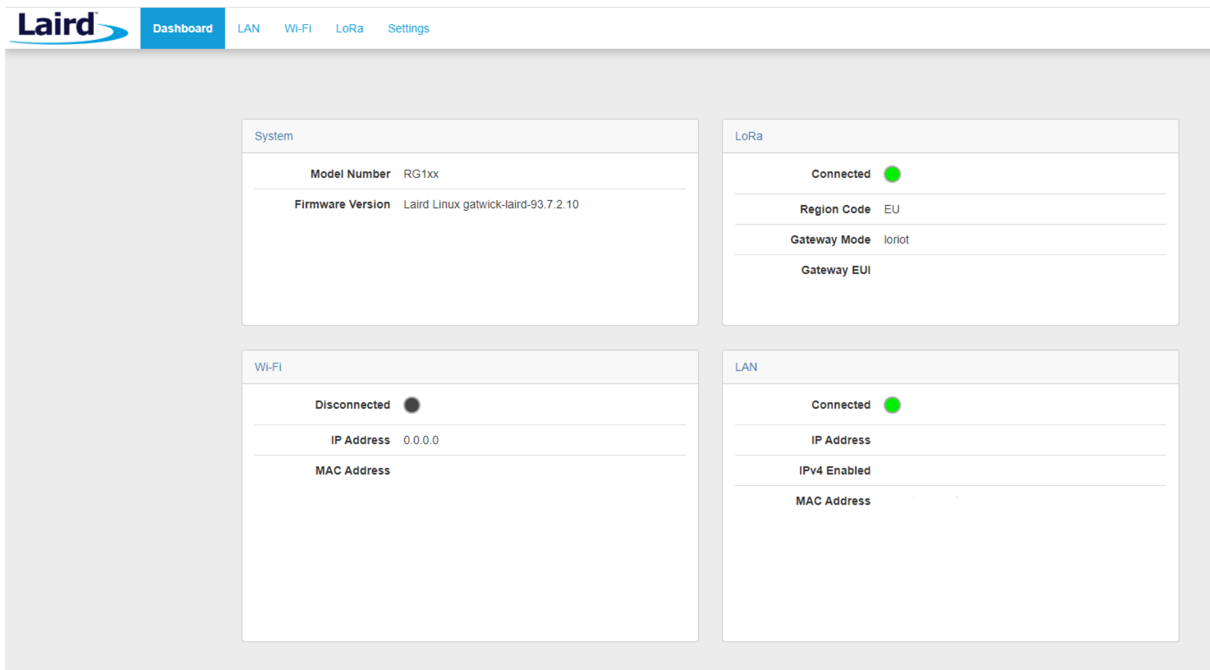

Figure 4-6 Laird Gateway Server Dashboard

LoRaWAN application is also set, and all needed data can be forwarded to be processed further. For this Application, HTTP Push was used to forward required data to Azure Function, which saves the data into a database. WebSocket and API tokens are required to be used for communicating the LoRaWAN Network server with the rest of the CI/CD system. These keys are used to deploy files from the automation server and verify the deployments.

4.2.2 LoRaWAN end device setup

This section presents the basic setup of the LoRaWAN node, a simple face recognition application and the modifications required to enable the CI/CD pipeline. In this section, the focus is on the setup's hands-on aspect. All basic Raspberry PI installation and setup are assumed to have been done earlier, so installing OS, the power supply, access control mechanism/actuators and actuators controlling software are not discussed as they are out of this project scope.

This setup was done on a Raspbian OS (version 5.10.52-v7+) installed into the Raspberry Pi 3B+. The setup and installation process requires internet access to install the needed libraries and clone the needed Git code. After the setup and node deployment, the node shall be updated through the LoRaWAN network. The IoT Node setup, in this case, has two primary aspects the LoRaWAN connection and the face recognition application.

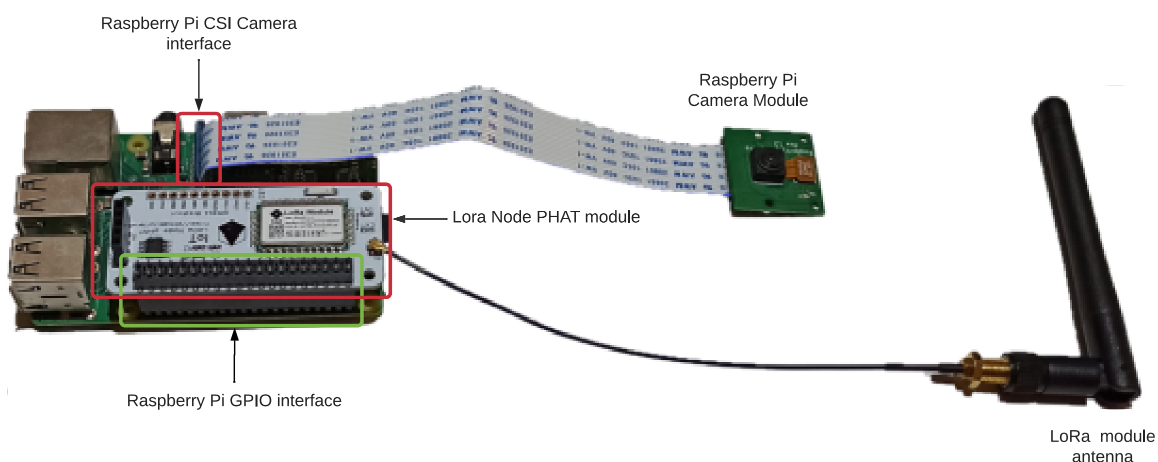

Figure 4-7 Raspberry Pi LoRa end device

Setting up the Raspberry PI end-device face-recognition application was made as to the following:

1. Mounting the LoRa Node PHAT module on the GPIO interface.

2. Mounting the Raspberry PI camera module on the CSI interface and enabling the Raspberry PI Camera interface using the command ***sudo raspi-config*** then navigating to Interface Options >> Camera >> Yes

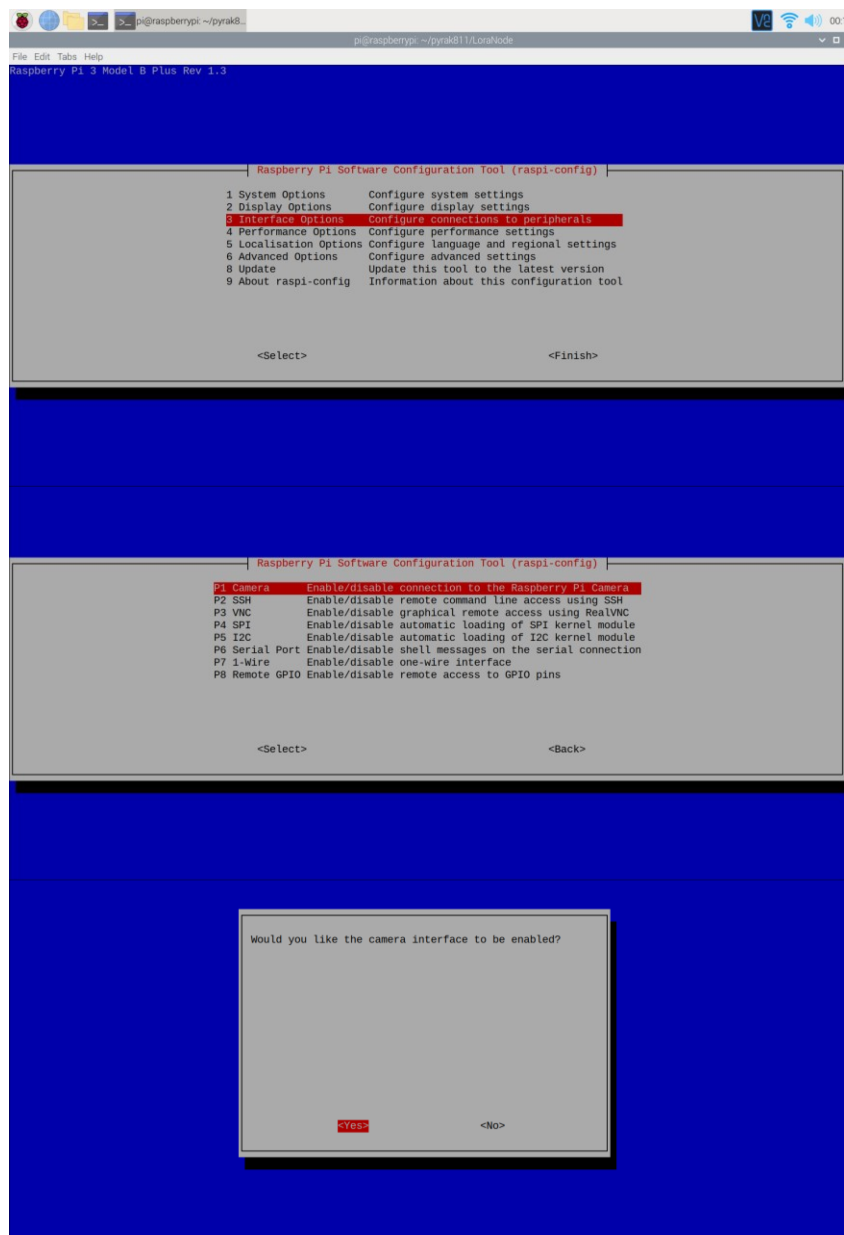

Figure 4-8 Enabling Raspberry Pi Camera interface

3. Installing the face_recognition library and the needed dependencies.
   sudo apt-get update
   sudo apt-get upgrade
   pip3 install face_recognition
   pip3 install opencv-python
   sudo apt-get install build-essential cmake gfortran git wget curl graphicsmagick libgraphicsmagick1-dev libatlas-base-dev libavcodec-dev libavformat-dev libboost-all-dev libgtk2.0-dev libjpeg-dev liblapack-dev libswscale-dev pkg-config python3-dev python3-numpy python3-pip python3-picamera

4. For the face_recognition application (GitHub repository, ageitgey/face_recognition), the example facerec_on_raspberry_pi.py was used with some minor modifications. This modification aims to use the saved reference image encoding file instead of the reference image itself to lower the amount of data transferred over the LoRaWAN with the remote deployment. Also, changes are needed to automate adding, removing, and recognizing reference encoding files. The face recognition code works by loading a reference image encoding file from local storage and pairing it with the person's name so the application can recognize the person and the name on the video or the picture input. The face recognition algorithms are out of this thesis scope and are not discussed here.

After setting up the face recognition application, the LoRaWAN node can be configured. This configuration is done for the RAK811 module used for this implementation:

1. Installing the RAK811 library using pip3 command ***sudo pip3 install rak811.*** The rak811 command can be used for modules with firmware V2.0.x, and rak811v3 can be used with modules with firmware V3.0.x.
   Python library rak811 is used for writing the program operating the module. After the installation, the LoRa node module can be configured through the command terminal. For example, ***rak811v3 hard-reset*** can be called to reset the RAK811 module.

2. The LoRaWAN node used for this project is written in Python. The node program configures the module parameters, joins the LoRaWAN network, sends a heartbeat uplink every 60 seconds, decodes the downlink messages, and saves the encoded image files received to the desired location on the local storage. After receiving the

file, the node also checks the file integrity by checking its checksum, comparing it to the file metadata received, and then sending the confirmation to the LoRaWAN server as an uplink. This confirmation uplink is needed in the CI/CD pipeline to ensure that files have been deployed successfully to the deployment target nodes.
First, all the needed connection parameters can be set, including the following parameters:

- LoRaWAN mode 0 or P2P mode 1: lora.set_config('lora:work_mode:0')
- LoRaWAN end-device Class 0:A or 1:B or 2:C: lora:set_config('lora:class:0')
- LoRaWAN joining mode OTAA:0 or ABP:1: lora.set_config('lora:join_mode:1')
- LoRaWAN region: lora.set_config('lora:region:EU868')
- Enable multicast: lora.set_config('lora:multicastenable:1')
- Set multicast address: lora.set_config('lora:multicast_dev_addr:{MULTICAST_DEV_ADDR}')
- ABP joining is used, which requires device address, application session and network session keys:
  - lora.set_config('lora:dev_addr:{DEV_ADDR}')
  - lora.set_config('lora:nwks_key:{NWKS_KEY}')
  - lora.set_config('lora:apps_key:{APPS_KEY}')

```
pi@raspberrypi:~/pyrak811/LoraNode $ python3 abp_v3.py
2022-07-27 18:28:10,332 INFO     Setup
2022-07-27 18:28:15,264 INFO         Work Mode: LoRaWAN
2022-07-27 18:28:15,265 INFO         Region: EU868
2022-07-27 18:28:15,266 INFO         MulticastEnable: false
2022-07-27 18:28:15,266 INFO         DutycycleEnable: false
2022-07-27 18:28:15,267 INFO         Send_repeat_cnt: 0
2022-07-27 18:28:15,267 INFO         Join_mode: ABP
2022-07-27 18:28:15,267 INFO         DevAddr: 004F6006
2022-07-27 18:28:15,268 INFO         AppsKey:
2022-07-27 18:28:15,268 INFO         NwksKey:
2022-07-27 18:28:15,269 INFO         Class: A
2022-07-27 18:28:15,269 INFO         Joined Network:false
2022-07-27 18:28:15,269 INFO         IsConfirm: unconfirm
2022-07-27 18:28:15,270 INFO         AdrEnable: true
2022-07-27 18:28:15,270 INFO         EnableRepeaterSupport: false
2022-07-27 18:28:15,271 INFO         RX2_CHANNEL_FREQUENCY: 869525000, RX2_CHANNEL_DR:0
2022-07-27 18:28:15,271 INFO         RX_WINDOW_DURATION: 3000ms
2022-07-27 18:28:15,272 INFO         RECEIVE_DELAY_1: 1000ms
2022-07-27 18:28:15,272 INFO         RECEIVE_DELAY_2: 2000ms
2022-07-27 18:28:15,272 INFO         JOIN_ACCEPT_DELAY_1: 5000ms
2022-07-27 18:28:15,273 INFO         JOIN_ACCEPT_DELAY_2: 6000ms
2022-07-27 18:28:15,273 INFO         Current Datarate: 5
2022-07-27 18:28:15,274 INFO         Primeval Datarate: 5
2022-07-27 18:28:15,274 INFO         ChannelsTxPower: 0
2022-07-27 18:28:15,274 INFO         UpLinkCounter: 0
2022-07-27 18:28:15,275 INFO         DownLinkCounter: 0
2022-07-27 18:28:15,275 INFO     Joining
2022-07-27 18:28:15,481 INFO     Sending packets every minute - Interrupt to cancel loop
2022-07-27 18:28:15,482 INFO     You can send downlinks from the LoRaWAN console or API
2022-07-27 18:28:15,483 INFO     Sending packet
2022-07-27 18:28:18,790 INFO     Amount of available downlinks: 0
2022-07-27 18:28:18,846 INFO     Sending packet
```

Figure 4-9 LoRaWAN end device serial output

After setting the parameters correctly, the end device can be enrolled on the LoRaWAN network on the LoRaWAN server. In Loriot.io, that can be done by navigating to the targeted application menu and Enroll Device option and adding the device parameters and keys.

Figure 4-10 Enrolling end devices into Loriot network server

Following a successful enrollment of the end device on the LoRaWAN server, the end device can join the LoRaWAN network and actively send and receive packets. The device status, statistics and messages can be monitored in the Loriot.io network server web interface.

Figure 4-11 LoRaWAN application activated end device

4.2.3 File Deployment over LoRa Network Server

A file receiving/saving mechanism is needed in the end device program. This mechanism needs to mirror how the CI/CD Server file fragmentation and deployment mechanism using the LoRaWAN server API calls, which implements a simplified basic FUOTA concept that does not follow all the LoRa Alliances recommendations as it is only developed as a proof of concept for this project. This File deployment application uses reserved ports 1,2 and 3 to send the file deployment path, file hash, and the total message count then use a different reserved port to send the fragmented file downlinks as multicast or unicast. The node device needs to follow the same concept and receive file metadata in ports 1,2, and 3, followed by the fragmented file on the specific port for file fragments. After receiving the number of downlink messages stated on port 3 (total message count), the node defragments the received file fragments, composes the file, and saves it to the target deployment path (received on port 1). Finally, the file sha256 hash is checked and verified against the received file hash (received on port 2) and the status of the file is reported back to the LoRaWAN server and then to the CI/CD server over a WebSocket.

First, a simple ASP.net C# web application was developed to test fragmenting and deploying files using the API interface of the LoRaWAN server. Once the concept was verified, a .net core command line application was designed to run automatically on the CI/CD server. The command-line application works by passing a few arguments specifying the following parameters:

- Deployment options: Single or multiple file deployments.
- The file or the list of files to deploy.
- The target file deployment path.
- The deployment targeted end devices or multicast (Device ID/EUI)

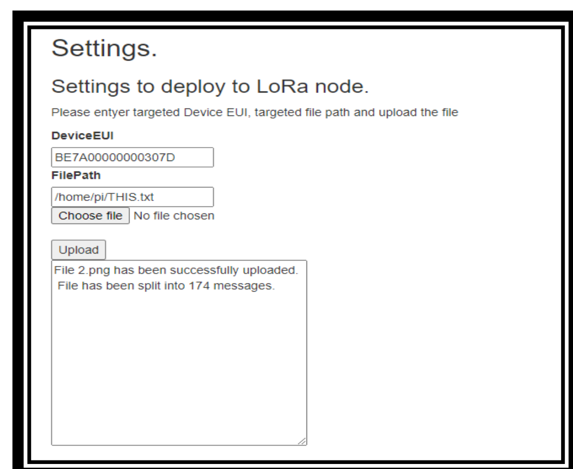

Figure 4-12 LoRa file deployment application

Sending files to end devices was done using the LoRaWAN server API interface. The API interface allows to send downlinks and enquire about the LoRaWAN network, gateways, and end devices. Few considerations were taken when designing and developing this file deployment software.

1. **The file fragment size:**
   As the LoRaWAN payload has a limitation of 222 B on SF7, there is a need to fragment files before sending the downlinks. The fragmentation can be decided according to the distance of the farthest end device, targeted by remote deployment, which dictates the DataRate and SF configuration. So, the further the most distant node within the LoRaWAN network, the smallest fragment size needs to be taken as lower DataRate can travel further. Using a smaller than required file fragment can result in too many redundant downlinks and causes redundant network traffic and higher energy consumption. On the other side, using a more significant fragment size results in the LoRaWAN network using the higher DataRate, which travels a shorter distance. The end device might not receive the downlinks if it is farther away.
   Further development is needed to optimize the fragment size according to the end-device RSSI of uplinks received earlier from the furthermost targeted end-device in the multicast group.

2. **The ability to deploy multiple files:**
   The software should handle multiple file deployment as there is a possibility of making changes to more than one file while developing node software or administrating access control.
   Multifile deployment is done by passing a file that includes the list of the deployment files. Each change to the repository automatically generates this list with the CI/CD server.

3. **The ability to deploy to multiple end devices:**
   The software should also deploy files into multiple device addresses or a multicast. The list of these devices shall be created automatically by comparing the latest available software version with the record of the latest deployed software to each end device or multicast. The targeted device list or multicast is passed as an argument to the software. It is automated by keeping a record of the deployed versions of the software and the latest access control list.

4. **The ability to verify the success of a file deployment:**
   The software should communicate with the end devices to receive deployment status from the end device. This communication is also essential to optimize the number of redundant downlinks sent, as the software only attempts to resend packets that were reported missing by the end device. The status verification contains the number of received packets, the amount and identifiers of missing fragments, and the file hash for verification.
   In this case, this communication is achieved by initiating a WebSocket connection with the LoRaWAN server, allowing the deployment software to access received uplink messages and verify or attempt to resend specific files or fragments.

5. **The ability to efficiently run the software automatically** through a script or a CD/CD pipeline.

A .NET Core command line application was developed to allow deploying multiple files targeting a unicast of multicast addresses. The application allows setting the target path of the deployed files, giving the tool more flexibility. Options were implemented as shown in *Figure 4-13*.

```
15      class Program
16      {
17          private static SHA256 Sha256 = SHA256.Create();
18          private static FileObject CurrentFile = new FileObject();
19          private static List<string> Devices = new List<string>();
20          private static List<FileObject> Files = new List<FileObject>();
21          private static readonly int MetadataPorts = 3;
22
23          [Verb("one", HelpText = "Send One file.")]
            3 references | moddon, 303 days ago | 1 author, 1 change
24          class OneOptions
25          {
26              [Option('f', "filename", Required = true, Default = "", HelpText = "A file to be deployed LoRa Device")]
                1 reference | moddon, 303 days ago | 1 author, 1 change
27              public string FileName { get; set; }
28
29              [Option('t', "targetpath", Required = false, Default = "", HelpText = "Target path in end device")]
                1 reference | moddon, 303 days ago | 1 author, 1 change
30              public string TargetPath { get; set; }
31
32              [Option('d', "Devices", Required = true, HelpText = "Input a txt file with the list of devices to deploy")]
                1 reference | moddon, 303 days ago | 1 author, 1 change
33              public string Devices { get; set; }
34          }
35
36          [Verb("multiple", HelpText = "Send Multiple file.")]
            3 references | moddon, 303 days ago | 1 author, 1 change
37          class MultipleOptions
38          {
39              [Option('f', "listoffiles", Required = true, Default = "", HelpText = "Input a txt file with the list of files to send")]
                2 references | moddon, 303 days ago | 1 author, 1 change
40              public string ListOfFiles { get; set; }
41
42              [Option('t', "targetpath", Required = false, Default = "", HelpText = "Target path in end device")]
                1 reference | moddon, 303 days ago | 1 author, 1 change
43              public string TargetPath { get; set; }
44
45              [Option('d', "Devices", Required = true, HelpText = "Input a txt file with the list of devices to deploy")]
                1 reference | moddon, 303 days ago | 1 author, 1 change
46              public string Devices { get; set; }
47          }
48
```

Figure 4-13 LoRaWAN file API deployer options

4.3 CI\CD Pipeline

CI/CD server is the heart of any CI/CD pipeline. In this case, a TeamCity server is used for automation, build management, and Continuous Integration to automate the pipelines discussed in this section. TeamCity is a highly customizable and configurable build automation system. The server relies on build agents to run automation tasks (builds, tests, deployments). So, for this project, as the targeted deployment platform was a Raspberry Pi, the build agents had to be hosted on Raspberry pi with an identical configuration (system image) as the end-nodes to achieve a stable build and test environment that is identical to the production environment. Besides the raspberry pi system image, no dedicated Configuration Management system was used. The task management system, application monitoring and feedback system, and databases are beyond this thesis scope. This section focuses on the automation of the testing and deployment tasks.

4.3.1 Face Recognition Application Development Pipeline

The pipeline illustrated in *Figure 4-14* makes it possible to continuously test and deploy the face recognition application's new features and updates to the end device. The software developers of the application are the primary user of this pipeline.

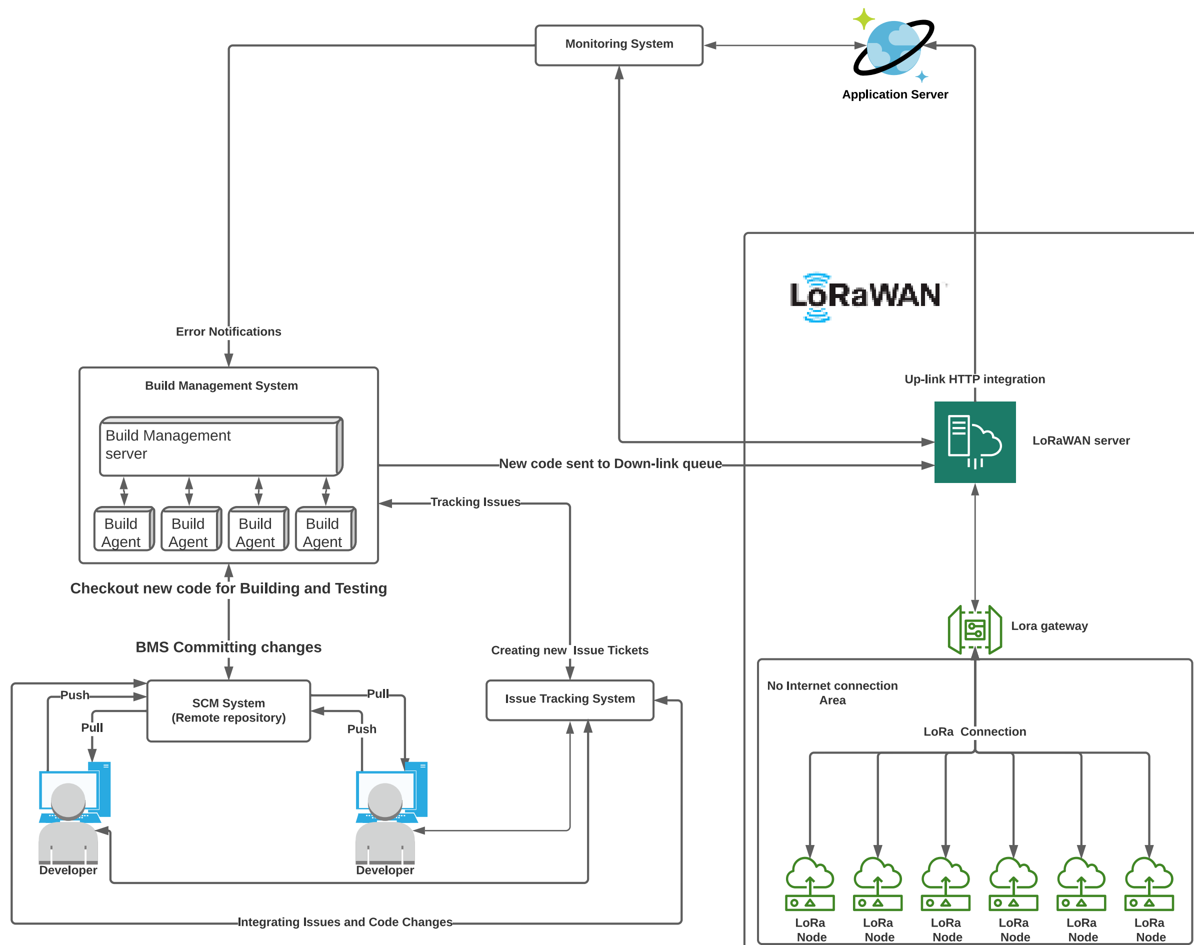

Figure 4-14 Face Recognition Application Pipeline

Application pipeline workflow is:

- Developers choose development tasks from the task management system.
- Developers write the new code and push it to the face_recognition repository feature or bugfix branch.
- Developers create a pull request to the main development branch.

- A TeamCity trigger is set to check out and trigger the build process (TeamCity Build Chain) with new code check-ins.
- If the build and the unit tests are successful, the pipeline runs the integration tests.
- When all the builds and tests configured to run on the development branch are successful, the developer is allowed to merge to the main branch.
- The main branch change triggers the tests, builds with the new code, and assigns a version number to the latest binaries.
- Successful integration tests on the main branch trigger the end device version check (saved to a local DB) to determine the list of devices targeted by the newer build version. Also, a list of changed files is prepared to pass to the file deployment software.
- The file deployment software then uses the list of devices that require an update and the list of files to be updated to do the fragmented deployment.
- End devices receive the fragments of the new files, defragment them into the original files, and then send a verification or resend request.
- File deployment software then confirms the updated devices and updates the local database with the versions of deployed software for each updated end device.

4.3.2 Images Encodings Deployment Pipeline

The image encodings update pipeline illustrated in Figure 4-15 allows the integration testing and deployment of reference face images to end devices. It serves access control system administrators to keep the reference images list up to date.

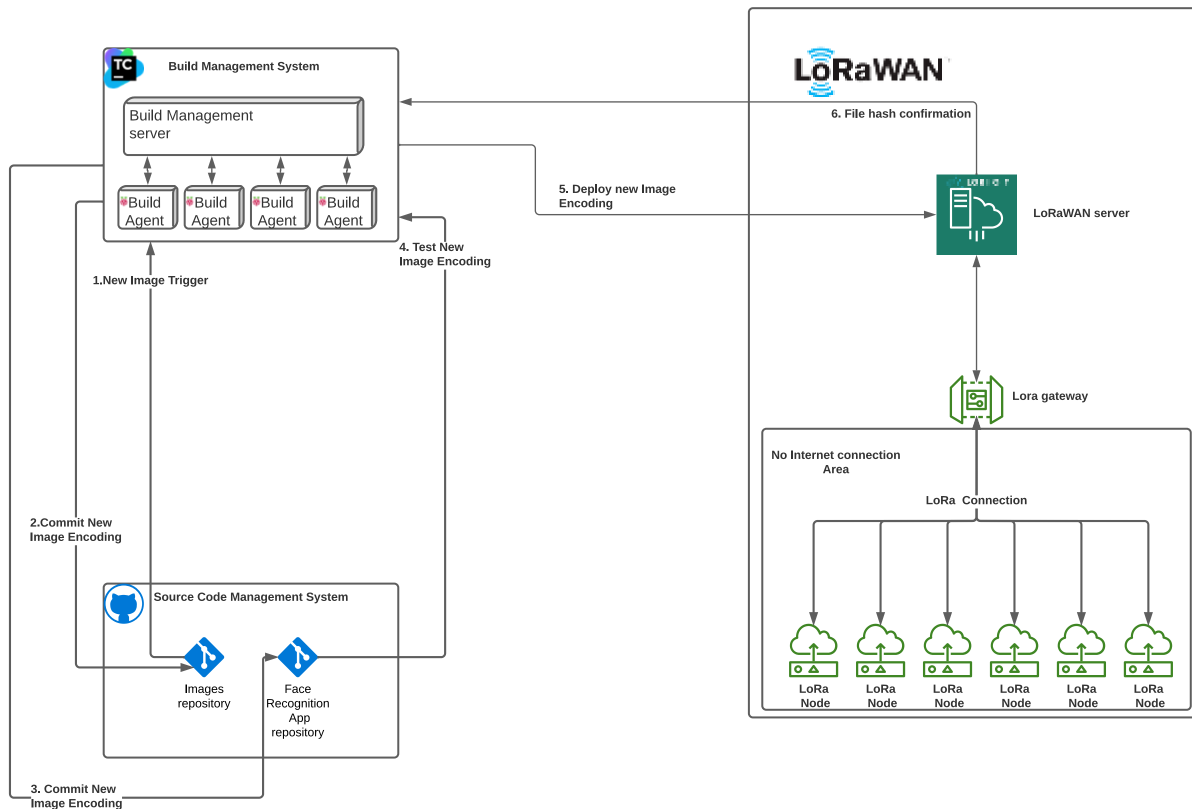

Figure 4-15 Images Deployment Pipeline (encodings)

The workflow of this pipeline requires a few modifications compared to the face recognition application Pipeline described in the previous section, so no unit testing is needed, as, in this pipeline, there is no code change to be tested. There is image reference file testing to verify that the reference files created are functioning as expected and can be used to recognize the targeted person. The reference images are saved into a central repository/cloud storage. Updating these images triggers the following chain of build configurations in the TeamCity server:

- Image encoding extracting. In this step, the build configuration uses the face_recognition library to load the image encoding and save the loaded array as a binary file. The usage of this array file reduces the amount of data deployed to the node as the array file is smaller than the original image.

- Test the encoding against test images. Adding new faces also requires adding a few test images of the same person to test the created encoding against the testing images and verify the functionality of the app and the validity of the extracted encoding.
- Update the encodings repo with the new encoding files.
- The updated encodings files are then pushed to a central encodings repository to be saved and for the changes to be traced.
- A TeamCity build configuration is triggered with the latest changes to assign a version to the latest change and create a list of the latest added encoding files.
- TeamCity runs a script that checks the latest encodings versions and timestamps in each end device which is tracked and recorded in a database. This software creates a list of end devices targeted for the encodings update.
- The File Deployment application then uses the new encodings files list and the list of end devices targeted to update. As in pipeline A, end devices then defragment the files and verify the files' reception status or require resending the files or specific fragments.
- File deployment software then confirms the updated devices and updates the local database with the versions of deployed encodings for each updated end device.

5 Results

This chapter aims to show the results of the system implantation described in chapter 4, designed to implement a CI\CD pipeline that can deploy code updates to end devices over a LoRaWAN network.

The CI\CD system implemented for this project was based TeamCity automation server to automate the deployment of new images' encodings and face recognition python scripts to the end device. The pipelines used Build Configuration Chain, where the tasks are triggered in series. Changes in the GitHub repository trigger the first Build Configuration (Face_Encodings_Generator) in the chain. When the build is successful, the build triggers the following build in the Build Configuration chain.

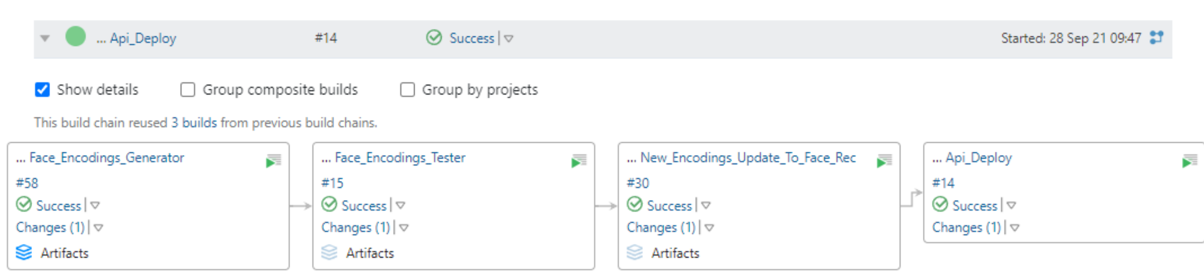

Figure 5-1 Build Configurations Chain

Figure 5-2 TeamCity Build Configurations

It was possible to track changes, set notifications and spot failures in the build chain. TeamCity Build Configurations provides a detailed log for each build step, as shown in the following figures.

Figure 5-3 Build Log view in TeamCity

Figure 5-4 Failed build errors

The end device has successfully received the encodings and python code updates with the DevOps pipeline implemented and deployment with LoRaWAN downlinks.
The Raspberry Pi end devices have been set to reboot after successful code deployments. The devices also have been configured to run the LoRaWAN node and Face recognition application at startup.

The pipeline improves the visibility and traceability of the changes done in both the image and face recognition repository. This visibility and traceability improvement helps decrease the system's recovery time or rolling back changes in case of failure.

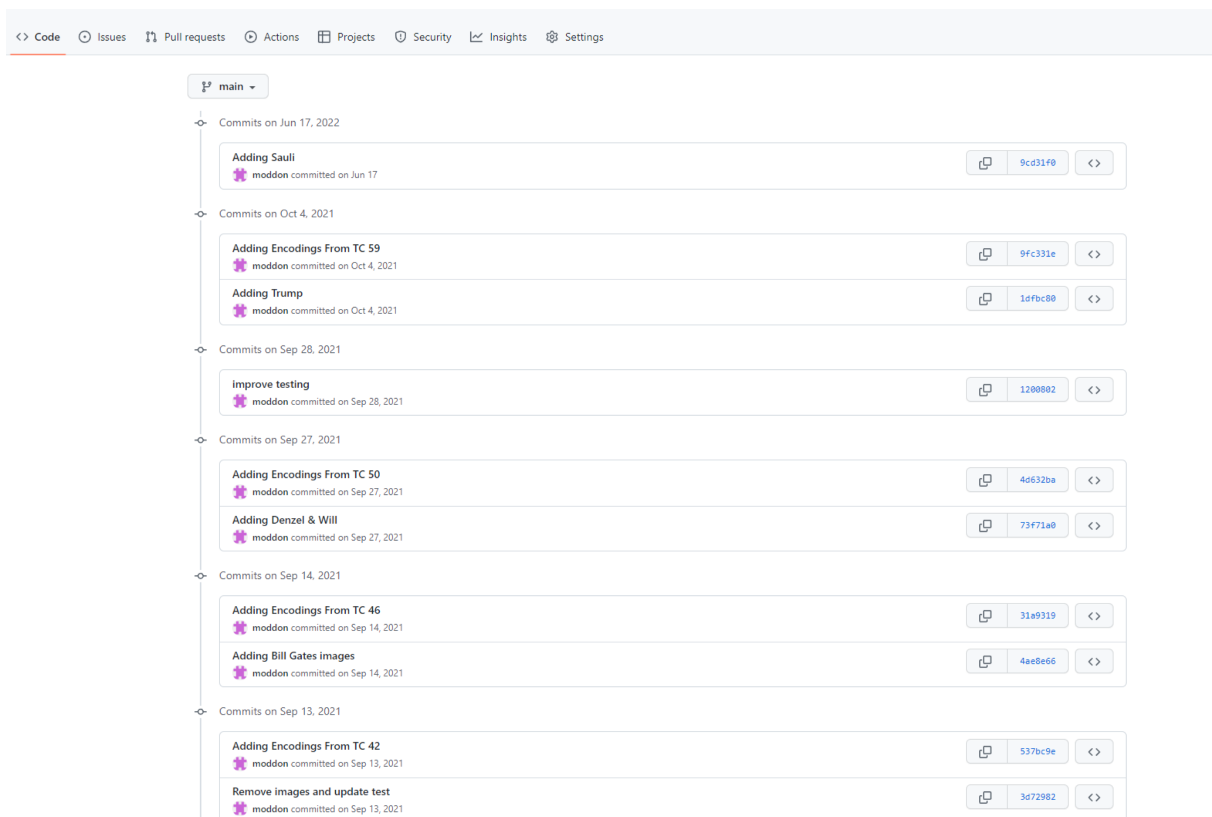

Figure 5-5 Repository Commit History

All needed uplinks downlinks data from the Loriot network server were forwarded using Lotiot.io Application HTTP Push to Azure function to collect the uplink and downlink data to improve the traceability of changes further. Overall, the system performed as expected, and the deployments were successful.

5.1 Potential Use Cases and Applications

We have designed and implemented a face detection application using LoRaWAN and a Raspberry Pi as an edge device. We have compared the following two systems to understand the potential use cases and further understand potential use applications.
System A: The system implemented in this thesis utilizes edge computing where LoRaWAN end nodes process the images and recognize the ones that have been referenced. There are 2 CI\CD pipelines in this system in this approach, one for the image encodings deployments and the other for the face recognition application new features and bug fixes, as shown in Figure 5-6.

System B: The need for remote LoRa node updates is avoided by utilizing central cloud computing integrated with the LoRa network server. The nodes send raw image encodings to the cloud to be processed and wait for the results to be sent as downlinks to do the action required, as shown in Figure 5-7.

System A: LoRaWAN Edge processing

This system has the same approach as the implementation in this thesis that utilizes CI\CD to apply changes to end devices over LoRaWAN.

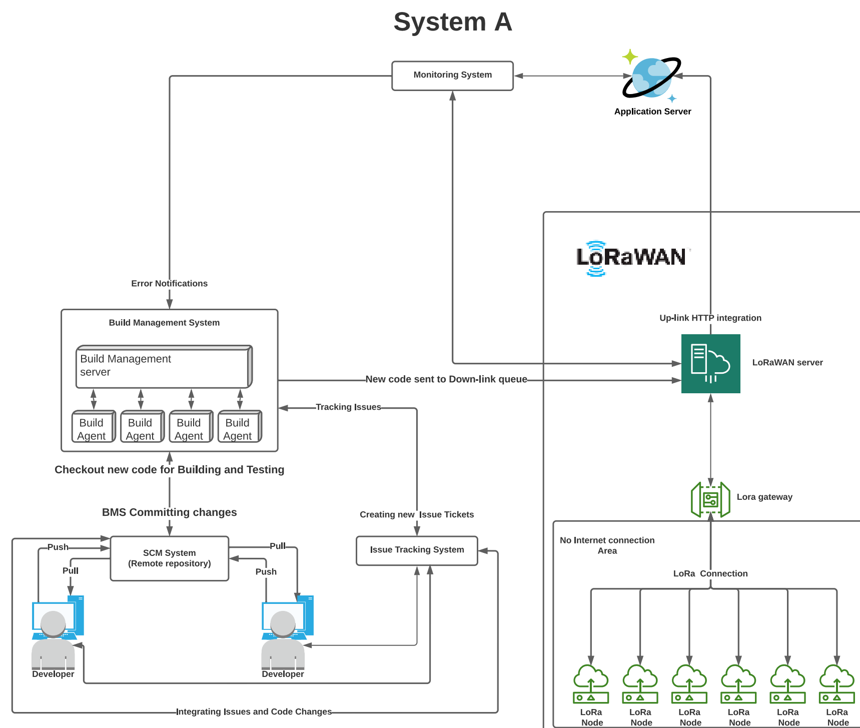

Figure 5-6 CI/CD for LoRaWAN Edge Computing

System B: LoRaWAN Central Cloud Processing

This system utilizes a central cloud processing (Web Application) to process the images and the access management. The system has a CI\CD pipeline continuously deploying changes to the could application where the image processing happens. The end devices do not do image processing. Instead, it sends the captured images to the web application to process and reply to whether a person is recognized or not.

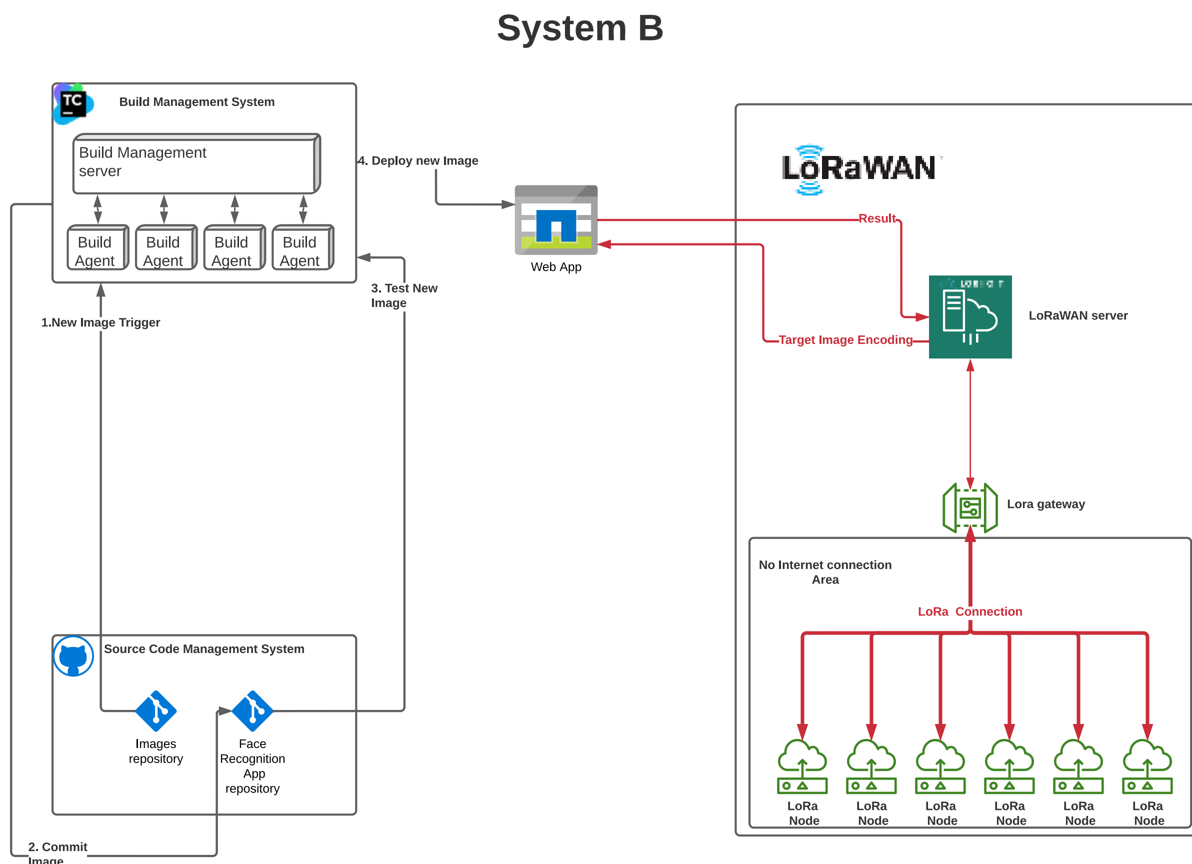

Figure 5-7 CI/CD for Cloud Computing

System B provides faster software deployment than System A as the deployment is not facing the same limitations introduced in System A due to deployment over LoRaWAN. On the other hand, the latency of transferring data over the LoRaWAN network affects the image processing time as every image captured needs to be encoded and fragmented in the node and then sent to the cloud server to be processed. The node must also wait for the cloud server's response with the results. This latency in image processing is a potential bottleneck for the system when increasing the number of images processed.

System A provides low latency image processing as the LoRaWAN nodes can process images locally. Latency of software or image encodings, on the other hand, is increased.

In *Table 5-1,* below are calculations of both systems' LoRaWAN packets amount. The calculations assume the following setup:

- The encoded image file size is 1 KB
- LoRa packet loss is 0%
- Applications packets are ignored
- Transmission is done with DR0 (51 Byte Packets)
- Packets needed to send one encoded image file is 1000/51 ≈ 20 packets
- Cloud computing image processing requires one downlink message per processed image to return the image recognition result to the LoRaWAN node in System B

Table 5-1 LoRaWAN Packets Calculations with Different System Scenarios

New images (encodings) / day	Images processed / day	System A		System B		Increase in packets with System B compared to System A
		Uplink / Day	Downlink / Day	Uplink / Day	Downlink / Day	
1	1	0	20	20	1	1
	100	0	20	2000	100	2080
	1000	0	20	20000	1000	20980
	10000	0	20	200000	10000	209980
10	1	0	200	20	1	-179
	100	0	200	2000	100	1900
	1000	0	200	20000	1000	20800
	10000	0	200	200000	10000	209800

From the calculations estimated in Table 5-1, there was a dramatic increase in operational LoRa traffic in System B, given that daily processed images are less than the number of new images that need to be updated daily. The only result that reflected better performance for System B was when the new daily images were more than the daily processed images.

The study of systems A and B shows that the implementation suggested in this thesis can have a noticeable advantage in reducing the LoRaWAN traffic and thereby reducing processing latency in various applications. Naturally, the design of the implementation suggested in this project could be extended to various time-critical edge computing applications. The design suggested in this thesis can be used in use cases such as access control in remote areas or large-scale deployments across industrial sites without the need for ubiquitous high-speed connectivity.

Just within the domain of computer vision, this thesis approach could be applied to plate recognition or other traffic systems, smart city management (e.g., monitoring of assets or automated alert systems) or large-scale monitoring in industrial applications.

Potential use cases could be extended to systems where equipment configurations need to be set remotely over LoRaWAN to gain more flexibility with the system.

6 Conclusion

The main goal of this work was to develop a reliable and efficient automated method for updating LoRa nodes' software, which was achieved by combining Continuous Integration and Continuous Deployment with the FUOTA concept. In this section, I go through the conclusion of this project.

When writing this thesis, we could not find a comparable end-to-end CI/CD documented solution implemented for LoRaWAN edge computing to compare the suggested implementation CI/CD performance or metrics. Earlier research focused on either DevOps implementation or Firmware updates over the air FUOTA. The work done in this thesis attempts to connect the research done in DevOps and FUOTA.

Although LoRaWAN has its limitations, the solution suggested in this work could benefit various use cases, as discussed in Chapter 5. This implementation can be improved by using adaptive LoRa date rates where an intelligent system can utilize Received Signal Strength Indicator RSSI to optimize the data fragmentation size and the transmitting data rate.

The use of hybrid communication can also be developed where end devices include higher-speed communication modules like NB-IoT that can be activated and deactivated remotely through the LoRa module, for example, when a more extensive file deployment is required.

References

[1] Gartner. (2022). Semiconductor industry revenue worldwide from 2012 to 2023 (in billion U.S. dollars). Statista. Statista Inc. Accessed: July 22, 2022. https://www.statista.com/statistics/272872/global-semiconductor-industry-revenue-forecast/

[2] WSTS. (2022). Integrated circuits semiconductor market size worldwide from 2009 to 2022 (in billion U.S. dollars). Statista. Statista Inc.. Accessed: July 22, 2022. https://www.statista.com/statistics/519456/forecast-of-worldwide-semiconductor-sales-of-integrated-circuits/

[3] IC Insights. (2022). Capital expenditure in the global semiconductor industry from 2000 to 2022 (in billion U.S. dollars). Statista. Statista Inc.. Accessed: July 22, 2022. https://www.statista.com/statistics/864897/worldwide-capital-spending-in-the-semiconductor-industry/

[4] IoT Analytics. (2018). Distribution of smart city IoT projects worldwide as of January 2018, by segment. Statista. Statista Inc.. Accessed: July 22, 2022. https://www.statista.com/statistics/869332/internet-of-things-smart-cities-projects-by-segment/

[5] Richter, F. (2021). Rise of the Robots. Statista. Statista Inc.. Accessed: July 23, 2022. https://www.statista.com/chart/26210/operational-stock-of-industrial-robots/

[6] Statista. (2021). Number of Smart Homes forecast in the World from 2017 to 2025 (in millions). Statista. Statista Inc.. Accessed: July 23, 2022. https://www.statista.com/forecasts/887613/number-of-smart-homes-in-the-smart-home-market-in-the-world

[7] Behmann, F, & Wu, K 2015, Collaborative Internet of Things (C-IoT): For Future Smart Connected Life and Business, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [July 23 2022].

[8] Stephens, R 2015, Beginning Software Engineering, John Wiley & Sons, Incorporated, Somerset. Available from: ProQuest Ebook Central. [July 18 2022].

[9] Flewelling, P 2018, The the Agile Developer's Handbook : Get More Value from Your Software Development: Get the Best Out of the Agile Methodology, Packt Publishing, Limited, Birmingham. Available from: ProQuest Ebook Central. [July 18 2022].

[10] Charilaou, C. et al. (2021) Firmware update using multiple gateways in lorawan networks. Sensors (Basel, Switzerland). [Online] 21 (19), 6488–.

[11] Khaled Abdelfadeel et al. (2020) How to make Firmware Updates over LoRaWAN Possible. arXiv.org.

[12] LoRa Alliance Technical Committee, (2019). FUOTA process summary technical recommendation. LoRa Alliance, Technical Recommendation TR002, January, p.v1

[13] Van Nieuwamerongen, S., 2021. Energy Consumption and Scalability of Transmitting Firmware Updates Over LoRa. Master of Science thesis. Delft University of Technology. Accessed: November 25, 2021. http://resolver.tudelft.nl/uuid:35f508c3-ab2c-4c69-b9f9-f1437d027404

[14] Lopez-Viana, R. et al. (2020) Continuous Delivery of Customized SaaS Edge Applications in Highly Distributed IoT Systems. IEEE internet of things journal. [Online] 7 (10), 10189–10199.

[15] Faustino, J. et al. (2022) DevOps benefits: A systematic literature review. Software, practice & experience. [Online]

[16] Verona, J., 2018. Practical DevOps. Packt Publishing.

[17] COUPLAND, M., 2021. DEVOPS ADOPTION STRATEGIES. [S.l.]: Packt Publishing.

[18] DevOps Research and Assessment (DORA) team at Google Cloud, 2021. Accelerate State of DevOps 2021.[Online]

[19] LoRa Alliance Technical Committee, (2017). LoRaWAN™ Specification Version 1.1. LoRa Alliance Technical Committee: Barcelona, Spain.

[20] LoRa Alliance Technical Committee, (2018). LoRaWAN Remote Multicast Setup Specification v1.0.0. LoRa Alliance

[21] The Things Network. 2022. Device Classes. [online] Available at: https://www.thethingsnetwork.org/docs/lorawan/classes [Accessed 12 September 2021].

[22] Semtech Corporation, (2015). AN1200.22 LoRa™ Modulation Basics

[23] LoRa Alliance Technical Committee, (2020). RP002-1.0.1 LoRaWAN Regional Parameters

[24] Semtech Corporation, (2020). LoRaWAN: Firmware Updates Over-the-Air

[25] Airtime calculator for LoRaWAN. 2022. [online] Available at: https://avbentem.github.io/airtime-calculator/ttn/eu868 [Accessed 26 February 2021].

Appendices

Appendix 1 list of figures

Figure 1-1 Waterfall Software Development Phases 6

Figure 2-1 Organisational and process introduced Wall of Confusion between Dev and Ops teams 10

Figure 2-2 Relationship between Velocity, Quality and Stability [17] 12

Figure 2-3 DevOps metrics value Dimensions suggested [18] 13

Figure 2-4 DevOps Process 14

Figure 2-5 DevOps Process Phases Association with Practices 17

Figure 3-1 LoRaWAN Network Architecture 21

Figure 3 2 Class A receive windows as appears in [21] 22

Figure 3 3 Class B receive windows as appears in [21] 22

Figure 3 4 Class C receive windows as appears in [21] 23

Figure 3 5 OTAA activation in LoRaWAN v1.1 24

Figure 3 6 LoRaWAN FUOTA network architecture as appears in [12] 26

Figure 3 7 FUOTA process as suggested in [24] 27

Figure 3 8 LoRaWAN Multicast 28

Figure 3 9 Multicast Control Package functions 29

Figure 3 10 End device parameters set by multicast setup 30

Figure 4 1 LoRaWAN Edge computing CI/CD Design 33

Figure 4 2 loriot.io network server geographical locations 34

Figure 4 3 Laird RG1 gateway Registration 35

Figure 4 4 Laird Gateway forwarder setting to Loriot.io 36

Figure 4 5 Loriot server registered gateway dashboard 36

Figure 4 6 Laird Gateway Server Dashboard 37

Figure 4 7 Raspberry Pi LoRa end device 38

Figure 4 8 Enabling Raspberry Pi Camera interface 39

Figure 4 9 LoRaWAN end device serial output 41

Figure 4 10 Enrolling end devices into Loriot network server 42

Figure 4 11 LoRaWAN application activated end device 42

Figure 4 12 LoRa file deployment application 43

Figure 4 13 LoRaWAN file API deployer options 45

Figure 4 14 Face Recognition Application Pipeline 47

Figure 4 15 Images Update Pipeline (encodings) 49

Figure 5 1 Build Configurations Chain 51

Figure 5 2 TeamCity Build Configurations 51

Figure 5 3 Build Log view in TeamCity 52
Figure 5 4 Failed build errors 52
Figure 5 5 Commit History in images repository 53
Figure 5 6 CI/CD for LoRaWAN Edge Computing 54
Figure 5 7 CI/CD with Cloud Computing 55

Appendix 2 list of tables

Table 2-1 Software Delivery performance results per performance category [18] 19
Table 3-1 LoRa DataRate parameters [23] 20
Table 5-1 LoRaWAN Packets Calculations with Different System Scenarios 56