MASTER'S THESIS 2023

Adaptive Synchronization and Orchestration: Tackling Offline and Intermittent Connectivity in IoT Environments

Gustav Engström, Victor Gunnarsson

ISSN 1650-2884
LU-CS-EX: 2023-79

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-79

Adaptive Synchronization and Orchestration: Tackling Offline and Intermittent Connectivity in IoT Environments

Adaptiv Synkronisering och Orkestrering: Hantering av sporadisk och avbruten uppkoppling i en IoT miljö

Gustav Engström, Victor Gunnarsson

Adaptive Synchronization and Orchestration: Tackling Offline and Intermittent Connectivity in IoT Environments

Gustav Engström
gu2025en-s@student.lth.se

Victor Gunnarsson
vi3851gu-s@student.lth.se

December 19, 2023

Master's thesis work carried out at Tetra Pak.

Supervisors: Roger Jönsson, roger.jonsson@tetrapak.com
Lars Bendix, lars.bendix@cs.lth.se

Examiner: Emelie Engström, emelie.engstrom@cs.lth.se

Abstract

Tetra Pak utilizes Azure IoT Edge for their configuration management and orchestration. Factories and machines are connected to IoT Edge devices that run IoT Runtimes with an EdgeAgent that orchestrates Tetra Pak's containerized software and handles configuration management. The configuration management is done through digital twin technology; each device has a twin stored in the cloud and one locally. When the cloud twin changes, the EdgeAgent detects this and applies the new configuration. However, transitioning to an IoT solution has introduced several challenges. Many customers operate in rural areas without internet connection, and some do not allow cloud connections. When customers do not have internet access, updating their IoT Edge devices is impossible. In order to solve this, Azure IoT Edge was analyzed to see if its capabilities could be extended. Despite some success, Microsoft's IoT Identity service hindered offline updates, as it needs a connection to the cloud in order to verify new identities. This led us to conclude that there was no way to support offline updates using Azure IoT Edge. As a result, we developed a requirements specification and a proof of concept of how configuration and container orchestration can be handled offline while maintaining the digital twin architecture.

Keywords: synchronization, digital twins, containerization, orchestration, IoT, IoT Edge, configuration management

Acknowledgements

First, We would like to thank Tetra Pak for the opportunity to work together.

We would also like to give a special thanks to our Tetra Pak supervisor, Roger Jönsson, for all the help and contributions in the discovery and prototype phase of our work.

Secondly, we would like to thank our university supervisor, Lars Bendix, for all the help in structuring the thesis work. We would especially like to thank Lars for all the useful feedback during the writing process.

Lastly, we would also like to thank Emelie Engström for taking on the role of examiner for this thesis.

Contents

Chapter 1
Introduction

In today's fast-moving world and business landscape, businesses must continually seek innovative ways to enhance efficiency and expand their value proposition. Tetra Pak provides high-end equipment for the beverage and food industry. They are now investing in digitalization and joining the worldwide wave of connectivity and the trend of IoT (Internet of Things). Further value can be gained from using IoT, such as continuous software delivery and continuous data streaming. And as artificial intelligence and machine learning continue to advance, all businesses that want to stay competitive need data to utilize the power of AI.

Tetra Pak has invested heavily in IoT and aims to further extend their competitive edge in the premium market segment by making their machines IoT-compatible as standard. A couple of years ago, they launched a pilot project to develop an IoT system for collecting data from various sensors. This led to collaboration between Tetra Pak and Microsoft. With this move, Tetra Pak gained access to a robust platform with great scalability and security.

With Tetra Paks IoT IoT-driven approach comes substantial benefits like scalable configuration management and data collection but also several challenges. One of those challenges is how to handle IoT devices with bad or no internet connection in terms of configuration management and orchestration of critical firmware. Many IoT systems utilize cloud systems to handle configuration and orchestration meaning that updates are not possible without an internet connection. Tetra Paks has many machines with intermittent or no internet connection. In the global food and beverage industry, facilities are often located in rural or remote areas without stable and fast internet access. Furthermore, some customers do not want Tetra Pak to access their data and prohibit Tetra Pak from connecting to their machines because some companies have strict policies on what data they share with third parties.

It is of great importance that machines remain functional even without an internet connection. Right now, it is less of an issue as the IoT infrastructure is only responsible for collecting data. Data collection is not considered a core part of the equipment. However, there are plans to include more critical software parts in the IoT runtime.

All IoT modules/applications run inside Microsoft Azure IoT Edge Runtime. The Edge Runtime is an environment that can run and orchestrate docker containers. This runtime

contains the Azure IoT EdgeAgent, a Microsoft-developed module that handles all the docker orchestration and the digital twin synchronization. This module does not support a continuous offline scenario; the configuration can only be changed when connected to the cloud.

The EdgeAgent utilizes a digital twin architecture for its synchronization with the cloud. A physical twin, the plant gateway, runs the EdgeAgent, which autonomously adjusts itself in response to its digital twin. The digital twin is a cloud twin that exists inside Azure's cloud storage. The twin is a JSON file that contains the configuration needed for all software running. The digital twin architecture will keep the digital and physical twin consistent. However, the exact moment of an update is unknown, but it will be performed as soon as conditions permit.

In the scenario of no internet connection, the EdgeAgent will not be able to download its digital twin from the cloud; therefore, it is not possible to perform updates to any software/modules that are running inside the IoT Runtime while the machine is offline.

If Tetra Pak's IoT solution is to be installed as a standard across all their machines, the issue with offline updates needs to be addressed. Tetra Pak needs to be able to deploy software in an offline scenario.

Through analyzing the open-source code and its mechanisms this thesis investigates how the Azure IoT Edge solution works. The analysis focuses on the behavior of the system within an offline scenario, mainly focusing on what mechanisms enable it to operate without an internet connection as long as there are no configuration changes.

Through this research, we aim to develop a requirements specification with what is needed to orchestrate and configure Tetra Paks software in an offline environment. This specification will focus on the key features and concepts needed to operate and support a machine within a disconnected setting. Furthermore, we will evaluate the most promising and viable concept moving forward based on the research on how IoT Edge works, the requirements specification, and prototyping.

Our work contributes to the field of configuration management and IoT by providing a unique offline perspective contrary to the normal use case of IoT. Currently, there is a limited amount of research on IoT in offline industrial environments. By addressing this gap, our research provides a further understanding of IoT applications in various operational conditions.

Chapter 2
Background

This chapter gives an in-depth background and context of the problem domain. We introduce the broader context of this study with insights into the existing landscape and environment in which the problems exist. The broader perspective is essential; it gives a comprehensive picture of the current state and knowledge and highlights the problematic areas to be investigated further in this work. We will identify the gaps in existing research and knowledge. In light of this, the research questions will gain their importance and frame the direction and purpose of this study.

The background will be structured as follows: Firstly, we will go into detail about the context of the problem. It will introduce the background of Tetra Pak and its customer landscape. Furthermore, the problem this paper revolves around will be presented as to why Tetra Pak has encountered this problem. The context of the problem will be tackled in the sense of what Tetra Pak seeks to gain from solving it.

Secondly, this chapter will present the problem statement of this paper and go into detail about why the need exists for the problem to be solved. Following this, we will discuss the method used to investigate the problem statement and go into the details of the pros and cons of the chosen method. Alternative methods will also be addressed as to why they were not utilized for this paper.

Lastly, a technical foundation will be presented to further the reader's knowledge and give an insight into the technologies used, described, or discussed within this paper.

2.1 Context & Problem

Tetra Pak is a global leader in food processing and packaging solutions, renowned for its pioneering work in aseptic packaging technology. The company designs and manufactures a wide range of packaging materials and processing equipment, vital in preserving and delivering liquid and semi-liquid food products to consumers safely, conveniently, and environmentally sustainable.

The food and beverage industry is a multifaceted landscape. It encompasses various products, from grains and fruits to dairy and processed foods. The industry is under constant evolution, with scientists all over the world constantly pushing for innovation. One of the most significant changes to the industry in the modern era is a growing awareness of the environment, sustainability, and health. This change drives consumer preferences towards more ethically produced food with a small ecological footprint.

Tetra Pak is at the front of this change and is investing a lot of resources in addressing these challenges and meeting customer demands. Their innovative food packages have revolutionized how food and beverages are packaged and processed. They have been able to extend shelf life without using preservatives. Something that has contributed to less waste and healthier food. Tetra Pak invests in research and development to reduce environmental impact.

Tetra Pak sells equipment for processing and packaging of liquid foods and is very prevalent in the milk industry. Due to the spacing needed for dairies, many facilities are situated in the countryside. Furthermore, these production facilities must also be close to the primary source of their raw material, cows, to reduce transportation costs and environmental impact. This geographic distribution creates challenges for Tetra Pak and its customers to overcome.

One of the biggest challenges in these remote settings is accessibility. Many of these dairy facilities can be challenging to reach. Consequently, delivering spare parts and technical support can take time and effort, and minimizing downtime is essential. As a result, Tetra Pak has put much effort into their connectivity. They are connecting entire factories through Internet of Things (IoT) technology, enabling support and updates to be done remotely.

However, the quality and availability of internet infrastructure could be better in many places in the world. Furthermore, some of their prominent clients prohibit Tetra Pak from connecting to their equipment. This is due to complex security measures and demands for adaptable and flexible equipment that they can integrate into their existing infrastructure and production. Tetra Pak strives to meet customer expectations and provides equipment tailored to their customer's needs. Therefore, new solutions and new features for their IoT platform are needed.

The current system relies on a network connection between the cloud and the physical device. However, to future-proof and extend the capabilities of the current systems, there exists a desire from Tetra Pak and its customers to be able to manage and update these physical devices even if they are offline.

In the current system that Tetra Pak utilizes, there is no support to perform updates without an internet connection. This poses a critical challenge that needs to be addressed for Tetra Pak to support their customers where reliable internet connectivity does not exist, is not allowed, or is intermittent. Furthermore, Tetra Pak has plans to extend the responsibility of the current IoT system, thus increasing the demand for a solution for managing offline devices.

The challenge lies in finding a solution to perform these updates offline while guaranteeing consistent versioning and configuration management. The solution must be compatible, integrated into the current digital twin architecture, and streamlined into the current systems and architecture. Furthermore, it must be secure and incorporate the safety mechanisms from Azure IoT Edge to safeguard the system.

Tetra Pak requires a solution that can perform updates in disconnected environments while maintaining the consistency and reliability of the current Azure IoT Edge service.

Safety remains a paramount concern in a production environment, and the solution must ensure that updates and changes are encrypted and secure from unauthorized access.

To address the given problem, we should divide it into specific parts. The central challenge revolves around comprehending how configurations should be synchronized between remote machines and the cloud. We break down the main problem into three different parts. Firstly, in the context of Tetra Pak, we must understand the current system and its functionality, especially its lack of offline management support. Secondly, we need to identify the requirements for correct configuration management in an offline scenario and align them with Tetra Pak's requirements. Lastly, we need to understand if extending the current system is viable based on our defined requirements.

The main research question that needs to be answered is:

How can synchronization of the digital and physical twin in offline and intermittent connection scenarios be supported in the current Tetra Pak context?

In order to answer the question, we need to look into different aspects; we have defined the following sub-questions.

(a) How does the Azure IoT Edge solution function in offline and online scenarios, and what are the core workings underlying its operation?

(b) What are the requirements specifications for ensuring coherence and maintaining synchronization and orchestration within the digital twin when operating in an offline scenario?

(c) Can the capabilities of Azure IoT Edge be extended to support updates in an offline environment? And if not, can we develop an in-house solution for managing synchronization and orchestration?

Question (a) is necessary to understand the problem and tools we have to work with. It is a complex system with many different interconnected components. Knowing how the system works under the particular circumstances relevant to Tetra Pak is crucial as Tetra Pak's systems and software rely on Microsoft's Azure IoT Edge solution.

Question (b) is essential to gain knowledge about what requirements are necessary for a system with responsibility for orchestration, synchronization, and version control. These requirements will become crucial for a future solution if the investigation deems it impossible to extend the current system's functionality.

Question (c) will evaluate if extending the current functionality to integrate offline support is possible. This involves investigating the feasibility of the features needed to allow the system to perform offline configuration updates. Moreover, this question involves exploring other viable solutions or ideas to solve the problem.

2.2 Method

This section exists to describe the framework and process that support our research. It analyzes different methods and their viability for answering the research questions. Moreover, it provides clear and detailed information about how our studies were conducted to ensure transparency of how we achieved our results. This allows readers to assess validity and reliability and offers insights into the scope and limitations of the research. Moreover, readers of this section who wish to build upon this research can get detailed information on our methodology.

To address the research question above, our methodology incorporates an iterative approach around theoretical and practical analysis, focusing on understanding the Tetra Pak context and their cloud suppliers' solutions. There are many different ways of doing this, and we want to focus on a holistic approach that aligns with the principles of action research. This approach involves continuous planning, actions, observations, and reflection which allows for constant adaptation and refinement of the strategies used. In the following table 2.1, we give an overview of the methods used to answer each research question, thereafter we go into more detail about why the approaches were chosen.

Table 2.1: An overview of the chosen methods in regards to each research question

Question	Methods
RQ (a) - Knowledge building through exploration	• Engage in hands-on experimentation. • Observation of the current systems. • Read and understand code/documentation.
RQ (b) – Specifying the Requirements	• Iteratively seek out and review the relevant literature. • Seek out and talk to industry experts. • Utilize the knowledge gained in RQ (a).
RQ (c) - Application & Evaluation of Extensibility	• Develop prototypes to test theoretical concepts. • Consult Microsoft. • Evaluate the prototypes according to RQ (b).

We must focus on Tetra Pak's needs and constraints, as many systems are already operating. Complementing literature studies with practical studies like coding and prototyping is essential. This is to understand the feasibility and viability of different approaches in Tetra Pak's context and unique industrial operations.

IoT systems are not made to work in offline settings by default, as the whole idea revolves around the internet. Most IoT systems are made to work in intermittent connection scenarios because the internet is not always available, even in developed countries. However, working in continuously offline scenarios with an IoT solution is not a rare use case specific to Tetra Pak's context. Therefore, to delve deeper into the current tools and answer our research question, the methodology of our thesis focuses on empirical and analytical research and understanding the Azure IoT Edge Platform and how it works in intermittent and continuous connection scenarios. Experiments will be conducted in the specific scenarios and environments relevant to our research. We will observe how the Azure IoT Edge platform works inside a simulated offline environment using test rigs in Tetra Pak's lab. This can help us identify the different components and mechanisms working together simultaneously during an offline scenario and identify the limitations of offline operations with the current systems.

Moreover, we will talk to industry experts to fully understand how the IoT Edge solutions work. This is to help us understand the practical challenges and solutions experienced by those working with Tetra Pak equipment and the current system.

From the experimentations and studies answering RQ(a), we can develop a requirements specification based on our studies, thus answering RQ(b). We will work through this iteratively, reviewing the requirements, and sort out the essential functionalities required. This will be done by meetings with stakeholders and technical experts. The requirements will focus on what is required for maintaining synchronization between digital twins in Tetra Pak's context and what is required for orchestrating the containerized software running on each edge device. This is important for the future development of this solution as a well-written requirement specification can act as a foundation for future development and research.

With the answers to both RQ(a) and RQ(b), we aim to make a prototype solution using the Azure IoT Edge solution and its mechanisms. We will investigate if performing an offline configuration update using the current solution is possible, thus answering RQ(c). Furthermore, depending on the results of RQ(c), we will consult Microsoft and see if they will add the functionality we need. To clearly articulate the specific requirements Tetra Pak has based on their context and the potential benefits that come with an integration of these features into the existing IoT Edge solution. Our approach will involve presenting a detailed approach of the requirements needed and what we think is the most promising solution moving forward based on our case studies of the Azure IoT Edge solution.

We will prototype a concept based on our requirements, showing how the configuration synchronization and orchestration can be done. This will be helpful because we can show a proof of concept to all stakeholders, which will analyze the feasibility of an in-house solution.

We considered other approaches to our chosen methodology, but most methods did not provide the desired holistic approach. Literature studies on their own can provide a solid foundation of how different concepts work, but they lack hands-on experimentation and direct observations based on Tetra Pak's context. Moreover, literature studies do not account for the current solutions and system in place, which a new solution requires to work alongside. Literature studies will be done iteratively as we gain a greater understanding of

the problem, and we will be able to find relevant literature on the subject. Literature can provide theoretical knowledge about synchronization, security, and cloud technologies.

On the other hand, Empirical research on its own is not enough either. It is a valuable tool in the development of a prototype or for understanding a particular solution. However, if we jump straight into prototyping before thoroughly analyzing the IoT Edge solution, we would not have the conceptual understanding needed in order to understand the broader concepts fully. We must consult the proper literature to gain critical insights and knowledge about best practices. Moreover, we do not want to reinvent the wheel; our solution should build upon existing knowledge and solutions and work in Tetra Pak's context alongside current software.

We aim to balance theoretical knowledge and practical application inside the Tetra Pak context. We will research iteratively, revisit relevant literature and documentation, and meet with competent people at Tetra Pak and Microsoft. Moreover, the prototyping will also be iterative, and we will work in sprints and show off our progression in terms of prototyping for the stakeholder continuously every two weeks. Findings for each phase will inform the subsequent one, ensuring that we center our research around the needs and context of Tetra Pak and fully understand the challenges of offline connection scenarios.

2.3 Theoretical foundation

This section is a key part of the report, as it provides the required knowledge for the reader to understand the problems and results presented in this report. In this section, we will go through some core concepts and techniques mentioned later. The following material will be described and presented:

- Docker
- Rust

2.3.1 Docker

Docker is a popular open-source platform that makes it possible to develop, ship, and run applications in a controlled environment. It makes it possible to handle infrastructure the same way as applications are managed. You can run applications inside Docker's isolated environment, which is called a container. This provides the security and isolation needed to run multiple containers simultaneously on a host. Each container is isolated from other containers and has everything needed to run. There is no need to rely on what is installed on the host. It is also very lightweight and can run on almost any operating system. It provides good support for continuous integration and continuous delivery.

Development can be done locally and then shared through a docker container, ensuring everyone has the same environment for executing the code. If a bug occurs, a new container can be developed in a test environment and then pushed and deployed in the production environment.

Moreover, docker can run in many different runtimes. In the context of Tetra Pak's IoT investment, they are running the containers on the IoT Edge Runtime on edge devices. Some more computation-heavy containers can be run inside the cloud environment and integrated seamlessly using the IoT tools provided by Microsoft. If more information is needed, please refer to the online documentation [3].

2.3.2 Rust

Rust is a very fast and memory-efficient programming language that has no runtime or garbage collector. It is optimal for use in stability- and performance-critical applications and running on embedded devices.

It has a unique type system and owner model which guarantees memory safety and thread safety. This makes it possible to eliminate many types of bugs in the compilation phase.

In the context of this thesis work, Rust is the language used in Microsoft IoT edge API, that runs in the background of the actual runtime on all edge devices. This API is open-source and part of the IoT Edge open-source project.

If more information is needed, please refer to the online documentation [15].

Chapter 3
Exploring Tetra Paks IoT management tools

This chapter explores the current system architecture Tetra Pak uses for its IoT management. To address the offline challenge, it is crucial to fully understand the current system and how it operates and utilize Azure IoT Edge. By examining the underlying mechanism Azure IoT Edge utilizes for online and offline synchronization and orchestration, we can answer the research question (a): *"How does the Azure IoT Edge solution function in offline and online scenarios, and what are the core workings underlying its operation?"*.

Firstly, the current systems and tools will be explored in detail. This will be done through reviewing internal documentation of their systems, meeting with experts, and analyzing the open-source code and documentation of Azure IoT Edge. This aims to create a general understanding of how Tetra Pak works with their system and what the different roles and responsibilities are for their systems. Secondly, this case study will, in great detail, explore the functionality of the Azure IoT Edge system. This part aims to give a comprehensive overview of the functionality, roles, and details of the different parts of the Azure IoT Edge system.

3.1 Tetra Pak tools and setup

In order to achieve scalability in their configuration management, Tetra Pak has developed the IoT Management tool, which is made for technicians and service personnel in order to facilitate and perform updates to Tetra Pak equipment. It utilizes Azure IoT Edge in the background and communicates with Azure APIs. Using the IoT Management tool, they can swiftly send critical software updates to all their machines and view what software and version is operating where.

This section explores the current system, concepts, and tools Tetra Pak utilizes in order to answer the research question (a). Firstly, the synchronization concept of their current IoT system will be discussed. This gives insight into how the current system works and introduces the concept of Digital Twins. Secondly, the IoT Management tool will be explored, and the role of this tool will be explained in depth. Furthermore, the tool's integration with Azure

systems will be discussed, as well as why this setup is necessary. Thirdly, Tetra Pak's factory setups will be explored in order to give insight into the requirements of the physical machines. Moreover, the term Plant and Plant Gateway will be introduced and explained. The software deployment process will lastly be discussed and shown in a figure to further explain how the current setup works in an online environment.

3.1.1 Synchronization within the IoT system

Tetra Pak utilizes a digital twin technology to orchestrate and manage the configuration of its devices. A digital twin is a virtual representation of a physical machine or software running on a remote device. The twin pair consists of two parts: First, the physical part, which is the machine located somewhere in the world, and the second part is a digital representation of the setup in the cloud.

Within the cloud environment, the digital twin is represented by a JSON file containing all necessary information regarding configuration. It holds data like environmental options, docker create and build options, as well as docker image versions and download locations. This digital representation also has a direct connection to the physical machine and can directly gather and synchronize information about the state of the machine, software versions, and data regarding sensors.

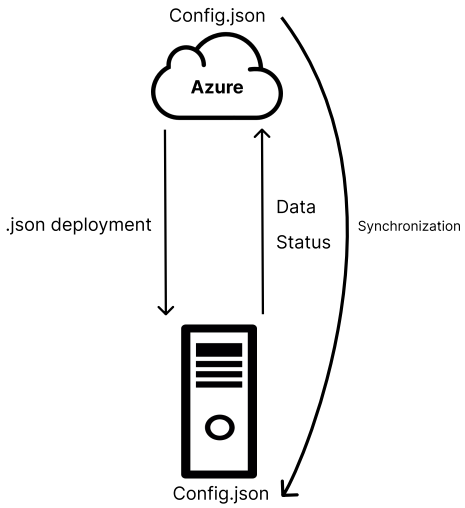

Figure 3.1: An overview of how the Synchronization concept works, what information is sent, and to where.

Using this concept, a factory can be managed remotely as long as it is connected to the internet. Furthermore, when a software update needs to be performed, this setup necessitates only updating the digital twin, as the system will eventually update the physical twin as it relies on the cloud twin for its configuration; this is visualized in the figure 3.1. The IoT Edge device will be consistent with the digital twin. However, the synchronization is not performed at a predetermined time because the system will only perform updates when it is considered appropriate. In an optimal scenario and in the current online scenario, all changes are made on the cloud, which are then propagated appropriately.

3.1.2 IoT Management Tool

Tetra Pak is currently operating, maintaining, and managing their connected factories with a self-developed tool called the IoT Management Tool. This tool allows for scalable operations in overviewing software deployments on multiple factories, but also for scalable deployments of software updates and changes. The reason for this tool's existence is that the Microsft Azure platform does not scale well with the operation and changeability of the IoT system, thus prompting the usage of other tools that can generalize the update process.

The IoT Management Tool's main purpose is to monitor and deploy updates to machines using the digital twin concept through Azure's API. The IoT Management tool performs updates on remote machines by rebuilding and changing digital twin representation within the Azure cloud. It utilizes a database hosted by Tetra Pak as data storage to gather the current setup and add on the changes. The database acts as the single source of truth, meaning the changes and updates stored in the database are, by definition, the setup that should be running on the remote machines. Furthermore, this data from the database should also be consistent with the setup representation of the digital twin in the Azure cloud.

The Tetra Pak IoT Management tool fetches the necessary data from the database, makes the respective changes to the twin file, and rebuilds the JSON file that is then deployed to the Azure cloud through the Azure API. Following this, the tool stores the respective updates to the database if a successful update of the cloud device is performed.

3.1.3 Factory Software Infrastructure

A factory is divided into different parts to isolate other parts of the production cycle for redundancy and security. Tetra Pak has designed their factory setups depending on the customer but also on the limitations and specifications of the Azure IoT Edge environment. A factory is described as having a tree-like structure, where the Plant Gateway is the root responsible for connecting the factory machines to the outside world; this structure can be seen in figure 3.2. Furthermore, individual pieces of equipment are connected internally to a local machine network, the production machines.

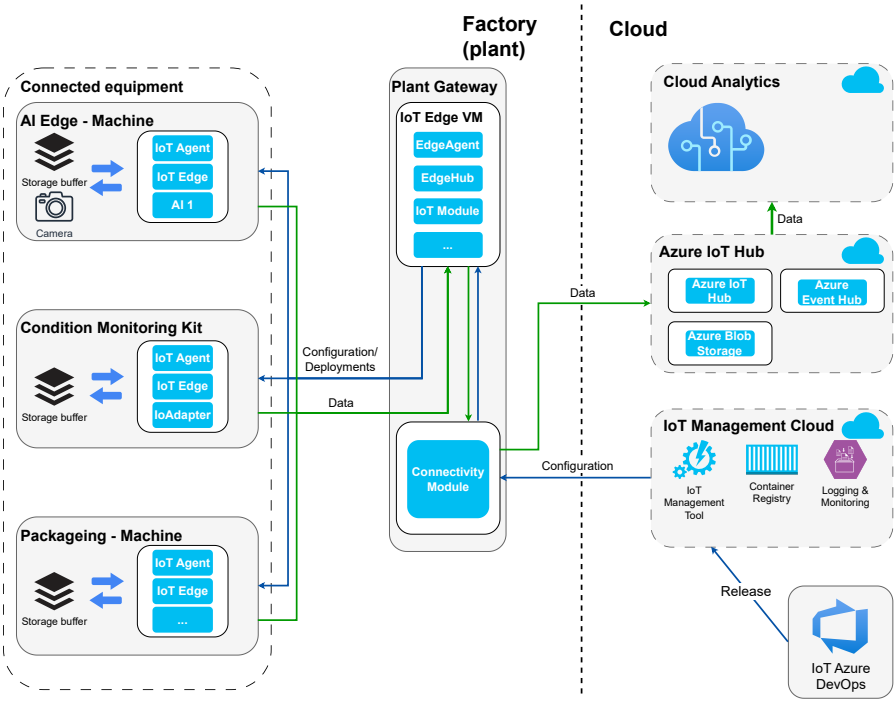

Figure 3.2: A visualization of a Factory (Plant) setup, including the connections to the cloud and local machines.

The Plant Gateway operates locally as a Windows-based industrial computer at individual customer sites. This gateway manages and monitors the other machines within the local factory network as an overseer and coordinator. Furthermore, the gateway acts as the cloud connection for all machines and bridges the necessary connection between the other devices on the network. In the figure 3.3 we visualize the machine setup and how it runs the IoT runtime in a Linux OS in a virtualized environment. The IoT Runtime starts a docker container for each module declared in the deployment.

Individual equipment, such as packaging, processing, or filling machines, are connected to the local factory network. These machines are configured differently from the Plant Gateway; the main difference is how they are connected to the network and that all leaf machines have the Plant Gateway as their parent machine. As mentioned above, this means that all communication to the cloud, including the fetching of digital twins, is passed through the Plant Gateway.

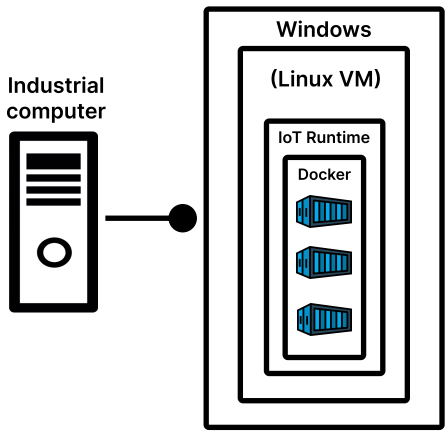

Figure 3.3: Visualization of how the Industrial computer (Edge device) runs the Windows operating system and then runs a virtualized VM that later runs the IoT Runtime and IoT Edge modules inside docker containers.

3.1.4 Software delivery process

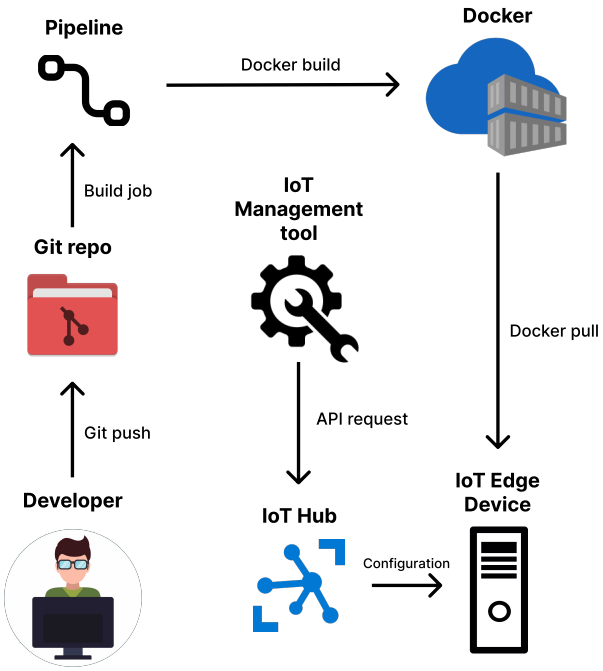

Figure 3.4: Flow diagram of online software update.

In the figure 3.4, we have made a flowchart of the software delivery process to make it easy to understand the flow from a developer's laptop onto a device. Firstly, the developer pushes new code to a cloud repository; then, their code is tested and built through a cloud pipeline. This build process results in a docker container pushed onto Tetra Pak's cloud container registry.

If a technician wants to update a device, they perform the task through the IoT Management tool. They select the IoT Edge Device that they want to update and what software they want to change/update. The IoT Management tool then requests IoT Hub to update the digital twin. The IoT Edge Device notices a change in its cloud twin when there is a new configuration. The IoT Device then pulls the specified container from the specified image, applies the configurations stored in the twin, and starts the new container.

3.2 Azure IoT Edge

Azure IoT Edge is a complete solution for providing distributed workloads on IoT devices provided by Microsoft. IoT Edge allows for managing, connecting, and communicating with IoT devices and allows custom workloads to be run in local containerized environments. A great benefit of the system is that it is entirely open-source and consists of several parts. It works with many kinds of hardware; they run it on a Linux VM in the Tetra Pak context. The Linux VM runs the IoT Runtime, which runs docker-compatible modules developed by Tetra Pak, Microsoft, or other open-source projects. Then, Microsoft provides the cloud infrastructure with an interface and API for managing and deploying workloads from the cloud.

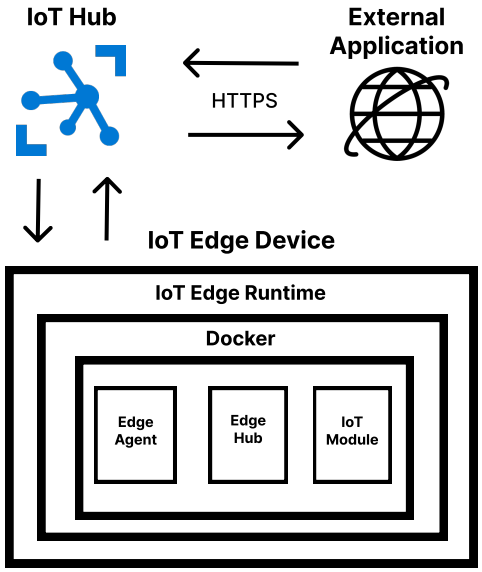

Figure 3.5: Visualization of the Azure IoT Edge system and its configuration.

3.2.1 IoT Hub

The IoT Hub is the cloud environment of the Azure IoT Edge system; see the figure 3.5. This environment contains the configuration and status information about every connected Edge device. The IoT Hub also provides the possibility to perform configuration changes and management of connected devices. Furthermore, the IoT Hub also acts as the access point for all devices to manage communication with the outside world, providing possibilities for outside communication through an open API.

3.2.2 IoT Runtime

The IoT runtime is the central runtime that executes all the modules running on each edge device; see figure 3.5. It uses Docker and the Moby-Engine in the background, and every module is a containerized instance, making the approach scalable and stable [13]. Furthermore, using Docker gives flexibility in the runtime and its capabilities. It has almost all of Docker's features and a toolbox complete with IoT and connectivity tools.

This high degree of flexibility is key for the IoT Runtime. Almost any application can be run inside a docker container, and all docker containers can be integrated into the IoT Runtime as modules. This allows all types of Tetra Pak applications to be migrated into this runtime as modules. This is an essential step for them to be able to orchestrate all modules using the EdgeAgent, which will be further elaborated on below.

In Tetra Pak's context, there are a lot of different modules running inside each runtime; there are some standard modules like the EdgeAgent and EdgeHub which are required to run in all IoT Runtimes as they are responsible for the core functionality of the Iot framework. Alongside these, many custom modules are running, which are developed in-house. Tetra Pak is investing much effort into containerizing modules, transitioning their software ecosystem to run inside the IoT Runtime and be orchestrated by the EdgeAgent. The possibility of combining containerization and connectivity is vital for increasing the functionality and scalability of their equipment.

3.2.3 IoT Edge Devices

Physical hardware is represented as an Edge Device in the context of the Microsoft IoT Edge. These devices are placed at the network's edges and connect the digital and physical environments [13]. In Tetra Pak context, the Plant Gateways are parent edge devices. The Plant Gateway's central and most important role today is to transmit the data collected from different parts of its network, orchestrate containers, and synchronize the digital twins. It is one of the critical parts of the IoT network. The Plant Gateway is the only part of the factory/site connected to the internet, meaning that all data transmission must go through it before it is sent to the cloud. As explained above, it is often an industrial Windows PC with limited processing power. However, it can utilize the powers of the Azure Framework and process some tasks on Microsoft-hosted servers with big processing powers.

The Edge devices are the host platform for the IoT Runtime, a core component that will be elaborated further in this report. This is important because the IoT Runtime executes and orchestrates all modules on each Edge device.

Although the edge device has limited processing power, it can perform simple calculations and data preprocessing. An example of this can be to organize data into blobs to regulate and optimize the required bandwidth. Furthermore, it supports localized computations, meaning it can perform some computations locally to reduce latency and remove the reliance on distant servers for production critical operations.

This makes it possible for each Edge Device to operate offline. However, an internet connection is necessary for configuration management and if the device relies on cloud computations for its production critical processes.

3.2.4 IoT Edge Module

As explained above, an IoT Edge module is a Docker container, a software component that can run inside the IoT Runtime environment. They are self-encapsulated inside a docker container, meaning the container is an entirely self-contained unit with everything the module needs to run: code, runtime, system tools, settings, and libraries. This gives portability and consistency for all modules [12].

Modules extend capabilities, allowing computation and data processing to happen at the network's edge and not only on centralized servers. In Tetra Pak's context, they have different modules for different tasks 3.2. Examples include machine learning, analytics, data transformation, and interfacing with sensors and devices. Furthermore, modules can interact with each other and the cloud seamlessly.

3.2.5 IoT EdgeAgent

The EdgeAgent is responsible for the software configuration management of other modules and orchestrates their versions and deployments. It can be seen as a manager of all other modules running on the device. The EdgeAgent, as seen in the figure 3.5, communicates with the IoT Hub to obtain deployment configurations, like which modules and their properties. The synchronization of the digital twins is eventually consistent as it syncs the runtime to its digital twin as soon as conditions permit [11, 13].

3.2.6 IoT Edge Hub

One module that is running on all IoT Runtimes is the EdgeHub, see figure 3.5, separate from the IoT Hub, which is the name of the Azure IoT cloud environment. It serves as a local message broker for all other modules on the IoT device. This is key for a secure and efficient data exchange between modules. This includes filtering, routing, and data processing before sending the data to its destination. EdgeHub supports many different protocols, for example, MQTT and AMQP, for its communication. This module is also responsible for buffering data in intermittent connection scenarios. It can queue messages when the network connection to the cloud is bad, ensuring no data gets lost[11, 13].

The EdgeHub is a crucial part of a decoupled network and a modular architecture. It enables the EdgeAgent to manage different modules independently without affecting each other, which is a critical feature in a production environment.

3.3 Offline Senario for IoT Edge

We have now explored the current system in an Online scenario. However, the system does work somewhat differently when offline. This chapter aims to provide an overview of the IoT Edge system when a device is offline.

If a Plant Gateway is offline, it cannot fetch its digital twin from the Azure cloud, instead, the device needs to use a local backup, which is visible in figure 3.6. The backup is a file stored on the Edge Device in a virtualized (VM) environment and then in the EdgeAgent docker container. The backup file, when stored, is encoded to BASE64 and then encrypted. The backup file is a JSON representation of the system containing information about the current system, such as the modules the system should have, versions of these modules, and other critical system information.

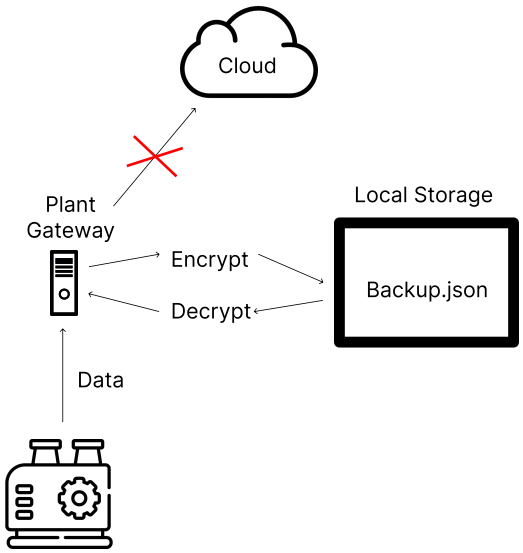

Figure 3.6: Overview of offline scenario.

It is essential to understand that when the EdgeAgent and the system are started without an internet connection, the EdgeAgent fetches the local backup, decrypts it, and uses the decrypted backup for its configuration. The actions of the EdgeAgent can be seen in figure 3.7, which depicts the scenario of encoding, encrypting, and storing but also decrypting and decoding.

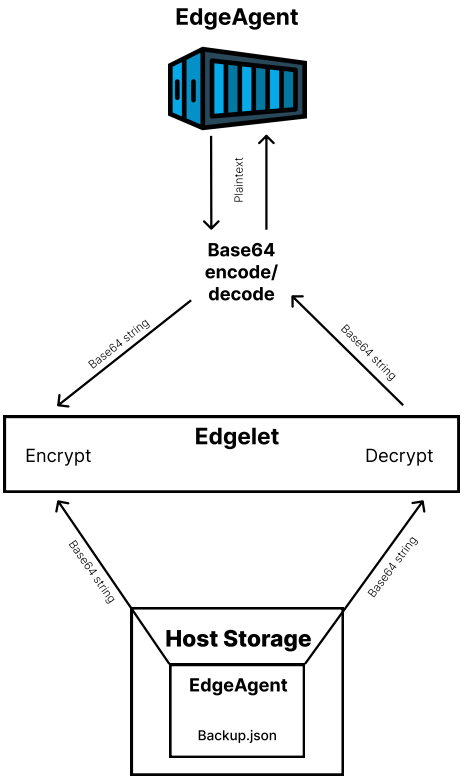

Figure 3.7: Detailed overview of backup, encryption/decryption.

The current backup file setup makes offline updates seemingly impossible; if a module version were to change and loaded into the system through Docker, it would revert the update at the next restart, as the version in the backup will not match the version of the module running on the system. Therefore, the change must also be registered in the backup file if an update is to be persistent when the system is offline. However, the backup file cannot be changed or read manually due to the file being encrypted for security reasons.

3.3.1 Azure Security Deamon

The Azure Security Deamon is the Edgelet that can be seen in 3.7 and in 3.8. This program is a local service that allows inter-service communication between modules and the system. The demon is split into different services, each allowing for different functionalities and capabilities. Each service is responsible for encrypting, decrypting messages, or signing and approving certificates. The Edglet services are all reached through a local Application Programming Interface (API). The API works the same as any other HTTP API; however, when run on Linux, the communication can only be done through .sock files. These files are specific to the Unix-based systems and function just the same as the localhost:<port> does in a Windows application. It is possible to communicate with these .sock files using programs such as Socat [5], which provide a bidirectional byte stream allowing messages to be sent and received or from within a created program.

The IoT Edge system uses these .sock files to facilitate communication between its containerized modules, as these provide a bridge between the containers and the runtime environment or Edglete. The sockets available for modules are the mgmt.sock and the workload.sock, these have different roles in the overall architecture of the API and facilitate different endpoints. In the figure 3.8, we can see the distribution of sock files for modules, but it also demonstrates the role of the Edgelet in the distribution of messages. According to the documentation [10, 13], the Azure security daemon has two different sockets for managing and distributing workloads. The workload sock is used as an open channel for all modules, contrary to the mgmt.sock, which is more for management purposes.

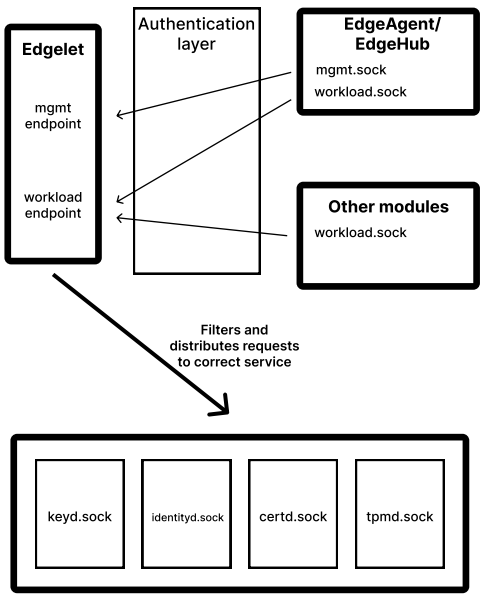

Figure 3.8: Overview of communication between Edglet, IoT modules, and the IoT edge services.

Several security features are built into the security demon that prohibits unauthorized messages from being applied to the system, and this is the Authentication layer seen in the figure 3.8. The security layer of the Azure Security Deamon is designed only to provide security through Process IDs or PIDs authentication. The Edglete, when starting, fetches all PIDs from all modules/docker containers it runs, then stores the PID information in an internal data structure where the PIDs are mapped to which module contains these processes, see figure 3.9. Then, when a module makes a request, it sends its module name, and the API iterates through the list of PIDs to check if the request is coming from an approved process.

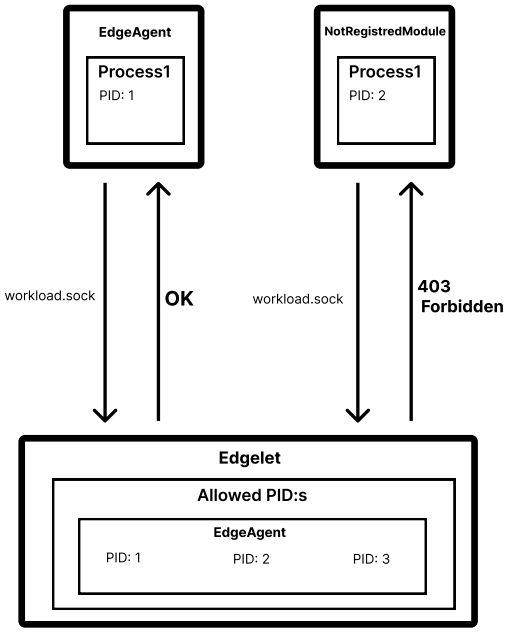

Figure 3.9: PID security overview.

Thus, in the figure 3.9, the NotRegistredModule will not be allowed to communicate with specific endpoints as mapping the internal PID of the module would not map to a registered module.

EdgeAgents usage of .sock files

The EdgeAgent has access to both the mgmt and the workload sock files to have complete access to the API [8]. This access allows the EdgeAgent to manage all modules through the mgmt.sock endpoints but also encrypt and decrypt its backup file through the workload.sock endpoints. In the Azure Security Deamon documentation [10], we discovered that the EdgeAgent encrypts its configuration file, which it fetches from the Azure IoT Hub cloud, each time it starts as a security backup if the system goes offline and performs a restart.

EdgeHub usage of .sock files

The EdgeHub has access to both .sockfiles [9]. However, the primary usage for the hub is to listen to the workload.sock as this is the message channel open to all other modules on the system. The hub then relays any message on the workload.sock to the correct route to other modules or the cloud.

Modules usage of .sock files

The main difference between the EdgeAgent, EdgeHub modules, and all other modules is that regular modules only have access to the workload.sock, see figure 3.8. The reason is that the workload sock file is the channel to facilitate communication between modules, the EdgeHub, and the cloud [11]. The EdgeHub then processes all messages sent from a module and are either relayed to another module via routes or sent to the cloud via the EdgeHub data route called the upstream, seen in figure 3.8.

3.3.2 Further exploration of the offline scenario

In this chapter, we explored the Tetra Pak Azure IoT system's inner workings in both online and offline environments and scenarios. This investigation is central to addressing our main research question: "How can synchronization of the digital and physical twin in offline and intermittent connection scenarios be supported in the current Tetra Pak context?" Our investigations have identified that an IoT Edge device remains functional in offline mode indefinitely. However, a significant limitation arose when we tried to update a container on the system; the inconsistency between the backup and container prohibits the system from operating.

Therefore, we propose a new idea to explore later in this report. Mainly, it revolves around changing/updating the backup file to get the EdgeAgent to run the update docker container. With this idea, we aim to test whether the system is flexible and extensible to support offline modifications.

The exploration of ideas sets the stage for the next chapter, where we will delve into the requirements. A requirements specification is critical to evaluate different concepts regarding feasibility and effectiveness. Our requirements specification will provide a structured framework, including the different requirements to achieve correct synchronization and orchestration. The following chapter will focus on developing a requirements specification to guide future exploration and ensure that the potential solutions align with system capabilities and the Tetra Pak context.

Chapter 4
Requirements specification

Designing a system that synchronizes documents, data, or files requires considering many aspects. In the case of software, it is necessary to consider additional requirements like dependencies, versioning, and the runtime environment. In order to synchronize any data, a system needs a source of truth, which is achievable with twin devices, one digital and one physical. The digital twin is typically a document containing information about the physical twins' setup, state, and version, in which the digital twin plays the crucial role of acting as the source of truth. However, syncing a digital twin with an actual device poses its challenges, particularly when internet connectivity between the two devices is not guaranteed.

This chapter will answer research question (b): *What are the requirements specifications for ensuring coherence and maintaining synchronization and orchestration within the digital twin when operating in an offline scenario?*, by reviewing and exploring Tetra Pak's requirements. We will explore the difficulties of information synchronization and orchestration with the twin devices, particularly in an offline scenario. Our requirements will build upon the insights from the previous chapter and explorations of the Azure IoT Edge system and how it works in the Tetra Pak context. The focus will initially be on outlining the general requirements for achieving coherent and synchronized data flow. These general requirements will come from literature studies on peer-reviewed configuration management papers found through literature searches.

In the second part, we will utilize the foundation concepts and formulate a requirements specification for an application that can solve Tetra Pak's specific issue. The Tetra Pak-specific requirements will be specified through meetings with experts and stakeholders along with the studies made during the previous chapter. This specification will not address the details but can serve as a framework and blueprint for implementing and evaluating solutions when investigating research question (c). By doing this, we aim to bridge the gap between theoretical concepts and research and practical implementation.

4.1 General requirements

In this section, we will analyze some of the more general concepts of configuration management with a specific overview of information synchronization. The aim is to create a general understanding of the synchronization issues and how the current research handles and solves these issues. These requirements will later give insight into what synchronization solutions the requirements of an offline orchestrator need to consider in the context of Tetra Pak.

When considering the synchronization of information between two devices, it is essential to recognize that changes must be tracked and kept consistent on both devices. If changes can occur at any time and on every device, conflict is highly likely. These conflicts may result in loss of information due to overrides or untracked changes. It may also result in inconsistencies between different devices, as they have locally stored versions of the information. Therefore, it is necessary to have proper version control and synchronization tools that can orchestrate, track, and manage any changes and propagate these to all devices.

4.1.1 Synchronization issues & solutions

In software development, a large portion of research focuses on exploring and managing information synchronization within organizations. The reason for this is that there is rarely a single person responsible for writing all the code, which results in several challenges regarding the constant synchronization of the code and frameworks.

This research area is called Configuration Management, often denoted as CM, which focuses on achieving systematic management of organizational/corporal configuration items. There can be several different types of configuration items; examples of this are hardware, documentation, or software. The latter of these is what we are focusing on in this master's thesis. With our configuration management system, we want to ensure that the identification and control of systems work throughout the software's lifecycle. The most important in our case is managing different versions of our configuration items (software) and maintaining consistency between different IoT Edge nodes. To help with this, research exists on best practices regarding version control.

Concurrency issues appear when multiple devices or users try to update the same information simultaneously [2]. Thus, we need a tool to guide these multiple users in performing changes to the shared information. In our scenario where our IoT Edge devices work under intermittent connection scenarios, there is a risk for concurrency issues in keeping consistency between the twin devices.

Suppose a device were to regain internet access during an offline update process. It could be disastrous and lead to a corrupt or overwritten system configuration. We need to establish a set of requirements and restrictions that will mitigate or eliminate these issues. One way of handling this concurrency challenge is to use the check-out/check-in model. This model can provide a structure for how and when to register a change. These actions ensure that changes are registered and stored safely.

Firstly, a document is checked out when a user or system wants to perform an update. This stores a copy of the document on their local system. This local version of the checked-out document is now only a clone and has no synchronization with the actual version of the document. Now, all changes can be performed on this document in complete isolation. After that, the updated document should be registered for check-in, which describes uploading

the document to the central storage location, which should maintain consistency between devices. However, there is the possibility that the document was checked out and checked in by another user or system, leading to a conflict. The system will try to re-establish consistency as follows: firstly, it tries to auto-merge, meaning if there are no changes to the same rows/functions/variables, it can choose the updated one. However, the user requires a manual merge if the conflicting files have different changes to the same part of the code because the system cannot decide what version of the code to use [4].

The check-out/check-in model is relevant to our work because it can be used offline; the person who wants to perform an update can check out the changes/configuration, of one machine, while he/she is online and later apply it to a device. If the system only allows one configuration/digital twin of a machine to be checked-out at a time, the system could safeguard against the concurrency issues caused by intermittent connection scenarios. Suppose an engineer checks out a configuration to be installed offline by the offline orchestrator. In that case, it cannot be checked-out by the online IoT solution until the offline orchestrator checks in the configuration, thus safeguarding against concurrency issues and guaranteeing consistency.

Using the check-out/check-in model could mitigate the issues regarding update concurrency of configurations, but it would likely not solve the issues regarding version control of individual programs. Fortunately, the individual programs at Tetra Pak are isolated into containers, which has its advantages. There are clear advantages to utilizing containers within a version control system [16]. The main advantage of containerization in the context of version control is that the different software versions are in isolated containers. Thus, a version control system can completely control a local machine environment. Containers would allow the system easy control of which container to run and, if necessary, roll back to an old container in the case of malfunction.

We can solve concurrency and version control issues based on the different concepts of check-out/check-in and containerization. Based on this foundation, we can formulate general requirements to help us solve our challenge. If we use these concepts, we can safeguard our system against concurrency issues by only making it possible to check-out a configuration once. Moreover, we can achieve version control inside our isolated environments using containerization. Thus, these concepts, supported by research, can be used as a base to build upon later in our specification.

4.2 Tetra Pak's Requirements

In the previous section, we explored the general requirements for synchronization and orchestration. However, to bridge the gap between theoretical concepts and practical solutions, we must adapt and tailor the requirements to work in Tetra Pak context. This is important because the conditions are particular to Tetra Pak and far from optimal. There is no connectivity and several security layers, and the solution must work seamlessly with the current solutions in place.

This section will present key aspects and requirements of a software application that will handle the synchronization and orchestration of Tetra Pak's distributed systems. The requirements are inspired by how the Azure IoT system works and what functionalities it provides. Furthermore, these requirements will consider the knowledge gained from the current research into software configuration management. This section will list the recognized requirements and, for each, give a brief description and discussion of why these requirements are necessary.

4.2.1 Stability

The application should run as a background service that is going to run on many Tetra Pak machines all over the world. The apps must keep running smoothly over time as they will be responsible for critical production equipment and software, any potential bugs can be dangerous and costly. Moreover, Tetra Pak is a company that works with food; therefore, food safety is paramount; potential bugs or instability can lead to dangers and costs.

To enhance and guarantee the stability of the orchestrator, it is vital to choose the correct programming language for the task. There are several other open-source orchestrators available which are widely used. Almost all are written in Rust or GO, considered the best and most modern languages available concerning performance, safety, concurrency, and low-level control. Example of this is Kubernetes and Docker Compose, which are written in GO. An example of an orchestrator written in Rust is the one Tetra Pak currently uses: IoT Edge Runtime. As Tetra Pak currently uses the IoT Runtime, Rust is the best language moving forward regarding compatibility.

Rust incorporates an ownership model for its memory management. It significantly reduces the likelihood of memory errors and concurrency bugs, some of the most common sources of system crashes and vulnerabilities in garbage collector languages.

To ensure stability, developers are recommended to use Rust for the software development. Rust's ownership model and compile-time error checking significantly reduce the likelihood of memory errors and concurrency bugs, which are common sources of system crashes and vulnerabilities in other programming languages. This feature aligns well with the need for a robust and reliable system in a food production environment.

Furthermore, it is a lightweight compiled language that offers performance comparable to C and C++ because it gives the developer control over system resources. As this orchestrator will run as a system service in the background, it must perform well and not take up too much resources on the computer.

4.2.2 Compatibility

The solution should be compatible with the IoT Management tool configurations and should be managed through the tool. All the files needed to perform the offline update should be downloaded while the field service technician is online and connected using the IoT Management tool.

Moreover, it must be compatible with Linux, as the host running the orchestrator is a Linux VM. As this is a system service, it will monitor many other services and the hardware. Rust provides excellent control for system-level software as it controls memory and system resources.

Lastly, this software must be compatible with docker as it will orchestrate docker containers. Docker uses an API for communication, and what is essential in regards to this is to have an application that can run specific tasks concurrently to reduce loads and increase performance. As Microsoft IoT Edge Edgelet is open source, the parts of the IoT Edge runtime that communicate with Docker can be of great use to analyze that code thoroughly regarding how they are doing their docker operations and how they have achieved such good performance and stability.

4.2.3 Single source of truth

Tetra Pak's IoT Management tool is the central configuration management tool for all machine configuration management. It is the only interface connected to Tetra Pak's cloud database containing global configurations for all machines. This centralization is essential to keep.

When changing the configuration for the offline orchestrator, the Tetra Pak IoT Management Tool must be the single source of truth. This is key for Tetra Pak to manage different machines and configurations. They must be able to see what modules and options are running on which machines in order to be able to act swiftly if bugs and issues occur.

When a configuration update is performed, it will be done through the IoT Management tool. This reduces the risk associated with direct manual interventions on machines and ensures a standardized update process, which only authorized personnel can do.

The IoT Management tool will be responsible for keeping the digital twin stored in the cloud up to date. As a field service engineer performs an update, these changes will be reflected simultaneously in the cloud.

4.2.4 Secure Docker Registry

The docker images should be hosted on the Tetra Pak cloud container registry; only containers from Tetra Pak's registry are allowed on machines. Therefore, a potential solution must only download files from the Tetra Pak registry.

This crucial security measure aligns with the best practices regarding container management. Tetra Pak has firewall configurations for all its factories and only allows requests to their server through the firewall. Furthermore, the network is split into several network levels, meaning some of the nodes inside the factory will not be directly connected to the Internet. Instead, they must download their configurations and container images from their parent node.

4.2.5 Orchestration

The solution needs to orchestrate the containers. Handle startup order of the different containers. It needs to handle restarts and updates to specific containers.

The offline orchestrator must have all the capabilities to orchestrate docker containers. This ranges from deployment to logs. The orchestrator must ensure that the containers are started in the correct order with the correct configuration. Ensure that the containers are restarted if they crash and automatically recover from failure.

Moreover, it needs to synchronize with its local digital twin and update its configuration if the digital twin changes. There will be a local digital twin if there is no internet connection, and it must make sure it is synchronized to this twin. However, it will be essential to synchronize at the right moment; no synchronization will be allowed while the machine is in production, potentially leading to data loss and production issues.

4.2.6 Logs

It must support logging vital information to enhance debugging capabilities and quickly discover and troubleshoot potential problems with the equipment and software.

This logging will be done offline, and a dashboard showing the logs would be helpful. A web application that can serve the field service engineer with data and diagnostics tools for analyzing the data.

4.2.7 Security

Protecting both Tetra Pak and the customer's data and integrity is essential. This could be done using a certificate chain of authentication, as only certified access should be possible.

When building a secure application for protecting Tetra Pak and the customer's data and integrity, selecting Rust as the programming language offers significant advantages, rust is renowned for emphasizing memory safety, effectively eliminating common vulnerabilities such as buffer overflows and dangling pointers through its unique ownership model. This aspect is crucial for an application dealing with sensitive information where such vulnerabilities could be severely exploited. Moreover, Rust's approach to concurrency ensures that data race conditions, often a source of complex bugs in multithreaded environments, are prevented, thereby enhancing the application's stability. Additionally, the absence of a garbage collector in Rust minimizes overhead and potential security risks associated with garbage collection, making the application more efficient and secure. With robust error handling and a strong focus on safety and security in its community and ecosystem, Rust is an ideal choice for developing a system that requires high security and reliability, perfectly aligning with stringent access control and data integrity.

4.2.8 Docker options

The tool must support docker create options as many containers need specific options. It is essential to have this kind of flexibility as this allows for customized deployments based on the unique set of requirements for each machine and environment. This is key as almost all of Tetra Pak's machines are custom-made and tailored to meet the needs of every client. If docker options are used correctly, the same container can be tailored to meet many different needs as it can be configured for each customer.

4.3 Summary

In this chapter, we have analyzed and discussed different requirements for an online and offline configuration orchestrator. As our main research question revolves around the synchronization of information and data, we analyzed the checkout/check-in model and software containerization. The analysis of these methods revealed that, while not adequate to solve the primary issue of offline synchronization fully, they offer groundwork in terms of understanding and knowledge that we can develop further.

Then, in the specific context of Tetra Pak, we discussed the overall requirements an orchestrator needs to follow to fit the context and environment. These requirements were identified and analyzed as a result of the in-depth overview of the Azure IoT Edge system and the Tetra Pak context and were formulated to fit the current system. The requirements for a potential solution focus on ensuring stability, compatibility, and security in a food safety environment. It must operate seamlessly and smoothly as a background service. The program must also ensure compatibility with IoT Management tools/solutions as a single source of truth. Moreover, it must be compatible with Linux and handle docker container orchestration in a stable and efficient manner. Additionally, It must support logging for troubleshooting and adhere to security measures to ensure a highly secure software solution.

Now that we have a clear set of requirements and a foundation of knowledge in configuration management and the issues that can occur, we can move on and try to answer the last research question (c): *"Can the capabilities of Azure IoT Edge be extended to support updates in an offline environment? And if not, can we develop an in-house solution for managing synchronization and orchestration?"*. Based on the requirements specification above and what we learned about Azure IoT Edge in the previous chapter, we can prototype and evaluate different ideas.

Chapter 5
Conceptualizing through Prototypes

This chapter will present our findings from the prototyping phase of this thesis work. This phase can be seen as a bridge between theoretical concepts and tangible solutions and plays a crucial role in exploring and validating our ideas, as often the devil is in the details. This exploration phase is not about creating a solution to an idea but about understanding, refining, and working iteratively with the ideas vital to answering the research question (c): *"Can the capabilities of Azure IoT Edge be extended to support updates in an offline environment? And if not, can we develop an in-house solution for managing synchronization and orchestration?"*.

The prototyping phase started with the analysis of the IoT Edge solution described in Chapter 3; in particular, the idea introduced in Chapter 3 was examined. Moreover, it is heavily influenced by the requirements we formulated in Chapter 4, as we have created and evaluated the concepts below based on the requirements and knowledge learned in the preceding chapters. The prototypes presented in this chapter enabled us to test our different hypotheses and conceptual understanding and discover the subtle details, obstacles, and requirements of the current systems. We explored and tried different ideas and their viability, firstly, we explored manipulating the current IoT Edge system, and secondly, we explored creating a self-developed solution. In this chapter, we aim to showcase the evolution of ideas, the challenges we faced, and the lessons we learned.

5.1 Manipuation of the local backup

As mentioned in Chapter 3, we discover how the IoT EdgeAgent utilizes a local backup file to store critical information during offline execution. Thus, changing information in this file was the original idea to explore. It was a great idea because if it were to work as we hoped, it would integrate seamlessly into the current solution. It is based on the fact that an IoT Edge device works great offline as long as there are no configuration changes. This means the localized backup on the host VM is utilized as it can recreate its containers without any internet connection, as shown in figure 5.1. Our idea was to access this local device twin,

change it from a field service engineer's laptop, and later sync that twin to the cloud from the field service technician's laptop onto the Azure cloud through the IoT management tool. The figure 5.1 provides a clear visualization of the scenario.

In this section, you can read about what we discovered and the different manipulations we could do to the localized backup. Moreover, we will go through the internal components of Azure IoT Edge and how they work based on what we found during our empirical research and examination of the code. This chapter aims to comprehensively investigate the described scenario to determine whether the IoT Edge system can support extension or if it is inherently inflexible.

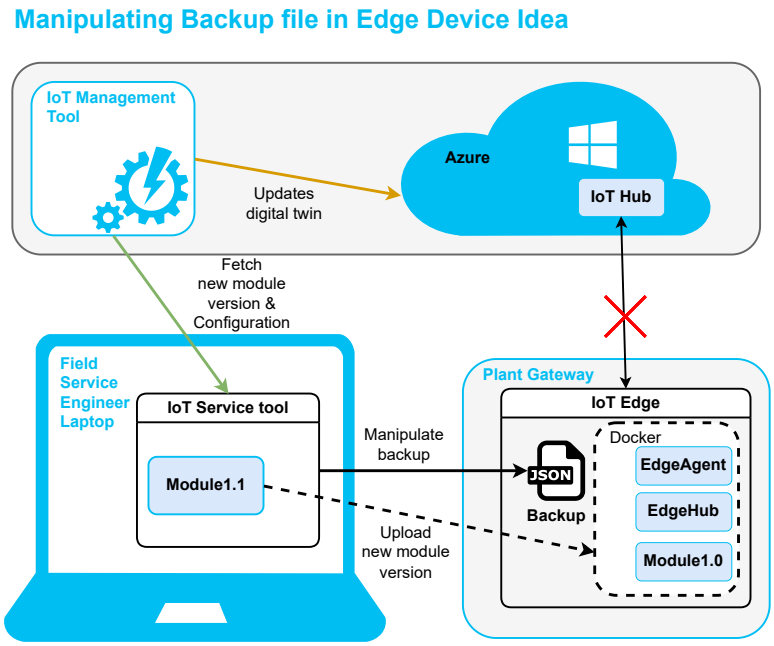

Figure 5.1: Visualization of the offline backup idea, where the Field Service Engineer (FSE) manipulates the backup file and loads a module container onto the Edge Device.

When digging deeper into the documentation and source code of the Azure IoT Edge application, with a focus on the EdgeAgent's role of managing Docker containers in an offline setting, it was discovered that the EdgeAgent is dependent on a backup file for persistent operations. This file serves as a backup copy of the critical information about the system and modules with associated Docker images. With the problem statement of handling offline configuration management in mind, the first design idea was to manipulate the backup file to change what software versions were deployed.

However, the first concern regarding the backup file was the encryption and storage within the EdgeAgents Docker container. A thorough review of the system logs, source code, and documentation revealed that the EdgeAgent relies on IoT Edge services to encrypt and decrypt the backup file. Furthermore, it was discovered that the EdgeAgent, during initialization and/or fetching updates, decrypts and encrypts its backup and stores it locally.

This dependency on the IoT Edge services posed a significant challenge in modifying the backup, as a non-encrypted backup file could not be read and loaded by the EdgeAgent. Due

to the built-in security of the IoT Edge services, it was impossible to access them without internal manipulation from within the EdgeAgent. Thus, to modify the backup file, it was necessary to get access to the internal services from outside the container.

5.1.1 Accessing the internal services

The interaction between IoT Edge modules, such as EdgeAgent and EdgeHub, and the IoT Edge API endpoints is accessed via the workload and mgmt sock files. These essential files are transferred to their respective Docker containers during their initialization, enabling direct communication between the modules and the security daemon. Furthermore, built into these endpoints is a security feature that provides a layer of security to limit which devices can access the API.

As can be seen in figure 3.9, Azure IoT Edge employs a security layer designed to return an HTTP 403 "Forbidden" when an unauthorized module sends a request to maintain security and prohibit unauthorized access. Authentication is done by looking at two parameters: the *module ID* where the request is coming from and the *Process ID* (PID) of the program making the request. Within the endpoint services, a map exists of which PIDs belong to which module ID, thus only allowing programs within a module access to the API. This authentication structure ensures that only those with access to the container's creation process, publication, and deployment can access the API.

Because we have full access to the build, creation, and deployment of the Docker containers in an **online** scenario, we managed to build a modified EdgeAgent with functionality that would provide outside access to the API. We deployed two socats within the EdgeAgents Docker container. These programs established a TCP connection with the internal workload and management endpoints, making the .sock files accessible on a host port. Using this setup, successful requests to decrypt and encrypt the backup were made. Thus allowing for modification and replacement of the backup file. This was confirmed by reviewing the EdgeAgent logs, which logged that the updated backup was successfully loaded after a restart in an offline environment.

However, it was clear that the system could recognize that a module configuration had changed. Subsequently, the system understood that the module needed a reconfiguration and prompted the system to update the registration of the module. Upon further investigation into why the updated module did not start, it was discovered that the registration of a module follows a specific sequence: The EdgeAgent makes the request to register a new module, and a specific request is sent to the management endpoint. This, in turn, calls a service called the identity service, which is a part of the IoT Edge service package. The identity service is responsible for provisioning and generating device identities within the Azure ecosystem.

Further investigation into this sequence revealed that during the module registration process performed by the EdgeAgent, the identity generated by the identity service needed further validation. According to the findings in the system logs, source code, and documentation [7], see the figure 5.2; this validation is done between the EdgeAgent and the Azure cloud. This verification process is crucial for only allowing validated modules to be started by the EdgeAgent and is a crucial part of the security feature of the system. Thus, the feature significantly limits the possibilities of further examination of the offline idea of changing the backup file, as it is impossible to access the Azure cloud without an internet connection.

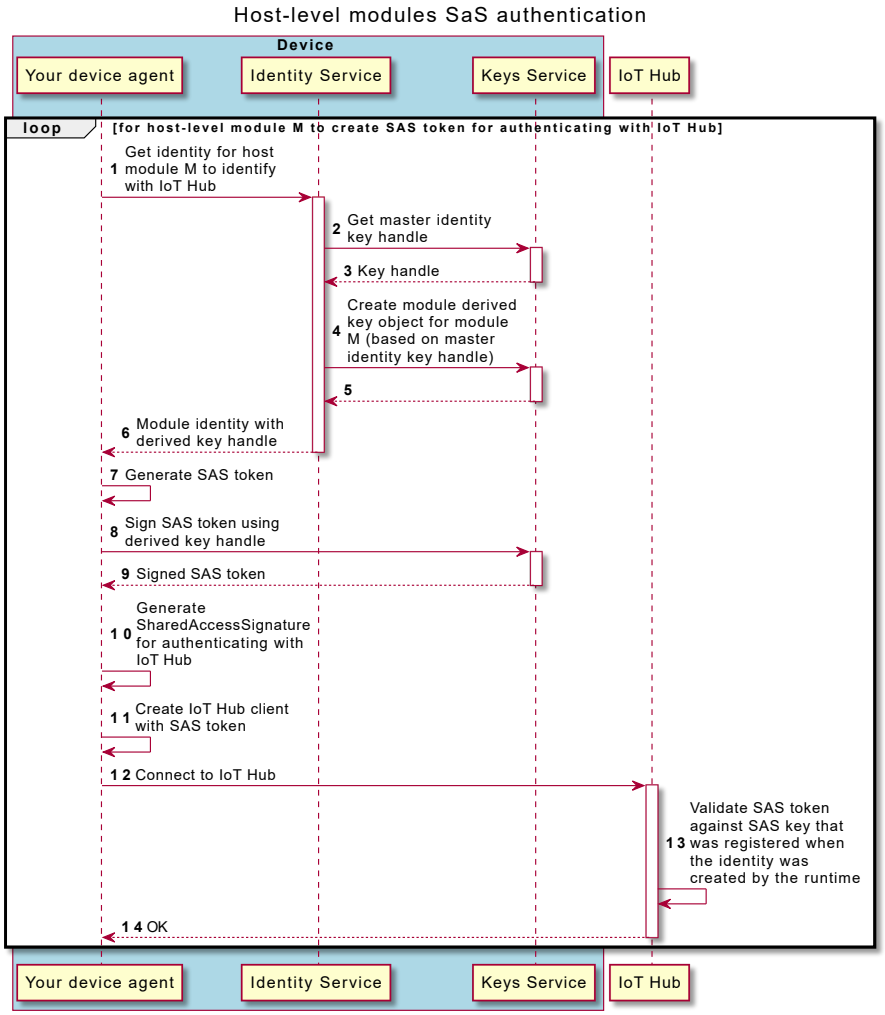

Figure 5.2: Image taken from the Identity service documentation, [7] demonstrating the flow of communication when authenticating a module.
`https://azure.github.io/iot-identity-service/develop-an-agent.html`

The requirement of having the EdgeAgent validate with the Azure cloud severely limits the possibility of creating a simple solution to the problem statement. However, two other ideas have the potential to solve this problem. As the EdgeAgent is open source, it is possible to rewrite this system to ignore or bypass the need for verification with the cloud. However, this is a comprehensive project and could introduce unwanted complexity and security impairments to the system. Thus, the other more feasible solution appeared to be more attractive. This solution proposed to emulate the verification process of the Azure cloud to act as a proxy for the actual cloud. However, as the verification process might be a Microsoft secret, this prompted a meeting with them to evaluate this solution's feasibility.

5.1.2 Meeting with Microsoft

At our meeting with Microsoft, we introduced the problem statement and the findings we had gathered from our exploration. We went through a detailed overview of how we ma-

nipulated the system to test our backup scenario and the problems with this approach. The spokesperson from Microsoft could not explicitly give us any concrete answers if the identification process could be emulated or disabled. However, they were supposed to continue the investigation and consult with more experienced and informed people.

Unfortunately, when we heard back from the Microsoft spokesperson, he explained that Microsoft does not offer any solution to our problem. However, it was suggested that we could use an IoT Module with access to the host system in order to track all changes and load the updated changes. However, Microsoft does not support this as it would not correctly update the device. Thus, we decided to stop investigating the problem any further. The only possibility for following up on this solution would be to rewrite the EdgeAgent, which is not within the scope of this paper.

5.2 Update module using the Edgelet API

This section will review a sub-idea we developed as we worked with the first idea; see figure 5.3. We listed this as its idea as it does not change the backup. Instead of changing the backup manually through the process above, we tried a different approach based on our understanding of the communication between the EdgeAgent and the Edgelet API. We tried to bypass the EdgeAgent and send the request directly to the Edgelet API. In this section, you can read about our insights from this investigation and why this is not a viable solution.

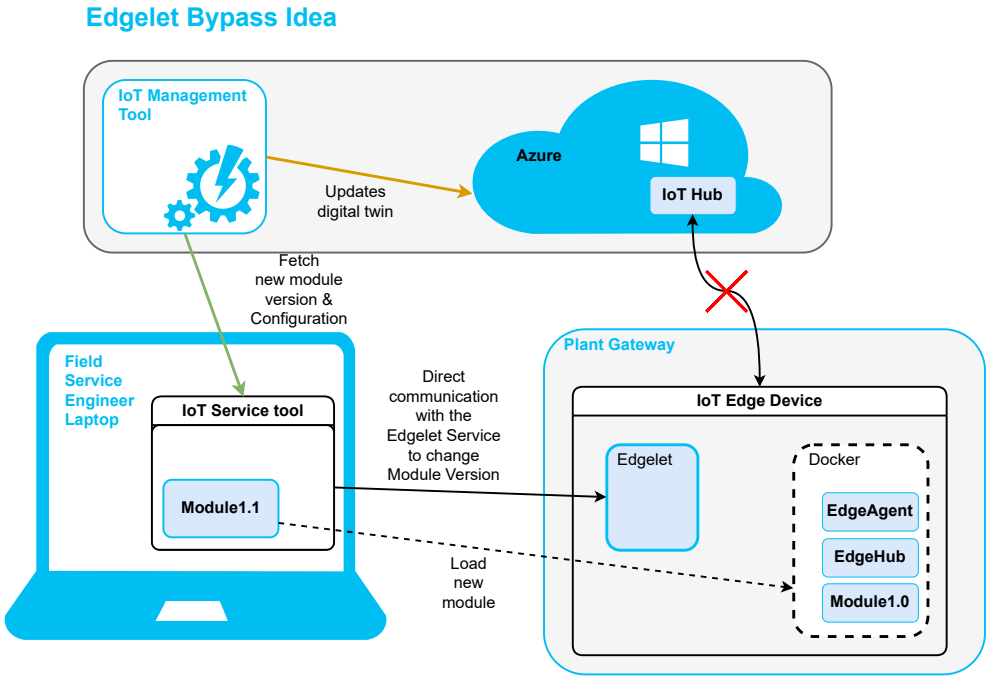

Figure 5.3: Visualization of the concept where we bypass the EdgeAgent, and communicate with the Edgelet instead.

During the discovery process of how the EdgeAgent uses the backup file, we discovered that the two primary endpoints, workload, and mgmt, act as orchestrators that, depending on the endpoint message, only forward the message to other services. These services have

different responsibilities, such as key, certificate, and identity management. The workload and mgmt endpoint's primary objective is to filter messages and to stop unauthorized access. The discovery of how Edgelet acts as a message broker and filter for the Edge Modules hinted at a workaround where it would be possible to work directly with the services through their APIs. This significant discovery prompted further investigation into the Edge Service and its functionality.

Reading the logs of the EdgeAgent made it clear that when a module is loaded, the EdgeAgent sends either a GET, PUT, or POST request to the management endpoint, depending on the state of the module. Thus, when trying to emulate the EdgeAgent, requests were sent to the same endpoint as discussed before, using socats. Furthermore, thanks to the documentation and open-source code, it was possible to understand the format of these requests and the expected responses. However, when we tried to emulate this, problems occurred when registering and updating a module (starting a local Docker image). Based on the logs and messages of the updated module, it was discovered that it was missing environment variables and files needed to execute and run it.

To further investigate this solution, the EdgeAgent was further explored, mainly how the EdgeAgent creates its messages for updating a module. From the gathered insights, more experimentation into emulating the requests was attempted. Eventually, these emulations proved to work when the request was properly constructed to include correct environment variables and files. The desired module finally ran with the correct docker image version. However, another problem was discovered. The module did not manage to start correctly; it crashed for unknown reasons. Further investigation into the module logs and the EdgeAgent source code gave no clue as to what made the module crash.

Consequently, this part of the scenario came to a stop in its investigation. Moreover, even if this scenario had worked, it would have needed further investigation into whether this solution would break and revert the updated module in the event of online connectivity or if it would maintain the updated and running module. This is because the management service's direct manipulation would not register the changes correctly in the EdgeAgent. Thus, further investigation would require registering the same changes in the EdgeAgent to keep the system consistent. Therefore this solution was not deemed as feasible as altering the system without notifying the EdgeAgent would introduce new problems, undermining the safety mechanism of the Azure IoT Edge solution.

5.3 Docker orchestrator prototype

In this section, we will explore the third idea, see figure 5.4, which is creating a prototype that will replace the IoT Edge Service in an offline scenario. As the ideas we have explored were not feasible, we implemented an offline orchestrator prototype. We took what we had learned about orchestration and synchronization and shifted focus toward building a working prototype to showcase how this could be done in a practical scenario. This prototype was heavily based on the requirements presented in Chapter 4, and we implemented the same synchronization concept as the IoT Edge solution with a digital twin representation but offline. The following sections will detail the prototype's architecture and explain the concepts and lessons learned throughout implementations and tests.

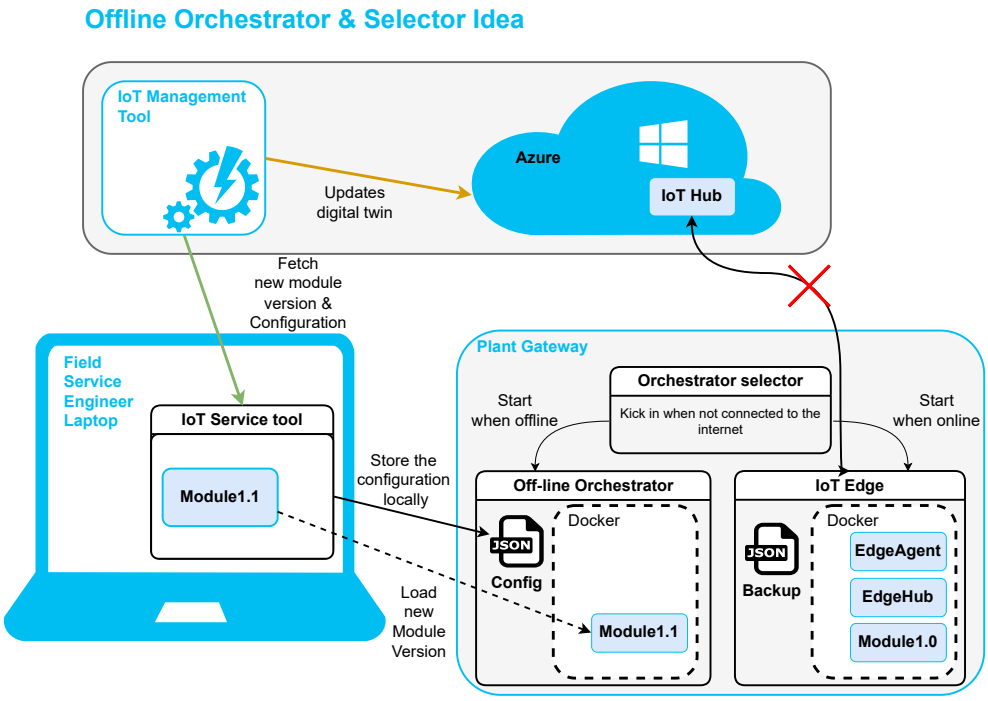

Figure 5.4: Visualization of the Orchestrator Selector and Offline Orchestrator idea, where the Selector changes what system will orchestrate workloads.

In Figure 5.4 there are two new pieces of software in addition to the current IoT Edge solution. These are the "Orchestrator Selector" and "Offline Orchestrator". The different orchestrators will be selected based on the connection scenario to guarantee seamless operation in intermittent connection scenarios. If the device is offline, the "Offline Orchestrator" will be set, and if the device goes online, it will choose the "IoT Edge." The offline orchestrator selector will handle this process. In continuous offline scenarios, the Orchestor selector can be disabled, and the Offline Orchestrator can be in charge indefinitely.

5.3.1 Supporting the Requirements

Based on the requirements defined in Chapter 4, and the lessons learned from exploring the IoT Edge System, we created a prototype for offline configuration management and orchestration. The system was created using the same programming language as the Edgelet system, Rust, to ensure stability and reliability. This programming language has many advantages and built-in functionality to ensure these requirements, as the language ensures memory and thread stability.

The plan was to write a minimal prototype that mimics the IoT Edge system while following the requirements defined in Chapter 4. This would allow the system to behave and function like the Edge system, ensuring compatibility with the current systems that Tetra Pak developed. Currently, the Edge system utilizes its ability to communicate with the local machines' Docker engine. This is done through Docker engines's exposed sock file distribution, which enables other programs to communicate with the engine through its API. Thus, the Offline Orchestrator can also interact with the Docker daemon in the same way as the IoT Edge system.

Furthermore, it was essential to be compatible with the digital twin infrastructure currently used by the IoT Management Tool and the IoT Edge system. This required our system to follow the same semantics as the current systems and provide the same parameters as the current systems. This digital twin configuration can be seen in the figure 5.5.

```
{
    "config_number": 1,
    "modules": [
      {
        "module_name": "",
        "container_name": "",
        "container_image": "",
        "restart_policy": "",
        "pull_policy": "",
        "startup_order": 1,
        "command" :"",
        "env_variables": {},
        "container_create_options": {
          "ports": {
            "8080/tcp": {}
          },
          "volumes": {
            "/module1/data": {}
          }
        }
      }
    ]
  }
```

Figure 5.5: Visualization of the config.json template utilized by the Offline Orchestrator.

The **configuration_version**: Specifies the version of the configuration that is currently loaded. This is to support both version tracking and also for supporting change control. Then, each module is wrapped in a list called **modules**. Each module is represented as a list of parameters like the IoT Edge Modules list, thus providing compatibility with these mod-

ules. The **moduel_name** describes the module's name; this will be essential to enable tracking which modules are running, and having the same name on the docker container as the module allows for easy mapping and understanding of what is running on a machine. The field **container_image** describes what container version a module is running. This field is one of the most critical fields as it describes which software version a module contains. Additionally, the configuration contains information about the particular docker container data to be inserted into the container on startup. These fields are **restart_policy**, the condition for when a container is restarted, and **pull_policy**, a policy describing whether a container image should be fetched from the local or an online registry at startup.

Moreover, the **start_order** describes in what order the module should be started. This can be important if different modules depend on each other and require one to start before the other. The start order does also matter in the case of a configuration change. The Offline Orchestrator is designed to restart all modules with a later start order than the updated one.

Further information essential to the docker containers are the **env_variables**, **command**, and **container_create_options**. The **env_variables** is a list containing all necessary environment variables to be added to a container. The **command** is a string containing a command that is run upon startup of the container. This field defines the entry point for a system, such as starting a particular program within the container or running a script. The **container_create_options** describes essential data for the container, such as which network ports should be open or which volumes the container should track on the host machine.

The configuration is highly inspired by the current configuration found in the IoT Edge system. To fully support synchronization at continuous offline operation, the system implemented a file listener for the configuration file. This listener can detect and register any changes to the configuration file and apply the new configuration.

The configuration is also essential, as this should be the file created in the IoT Management tool. Thus supporting the requirement for utilizing the IoT Management tool as the single source of truth of configuration creation. The configuration file will be the actual representation of the system and the one true copy that requires synchronization.

Further requirements defined in the earlier chapter regarding how the docker images are downloaded from the remote docker registry will not be handled by the Offline Orchestrator. This requirement would be more for the general IoT Service Tool, see figure 5.4, which has the network connection in an offline scenario. Then, the downloaded image can be loaded onto the Devices image registry with the help of the IoT Service Tool and later applied.

Lastly, security is vital for a system running business-critical software. This requirement can be met by utilizing the Rust language and its extensive security features. In Rust, memory utilized by programs is safely guarded due to the design of the ownership and borrowing system.

5.4 Summary

During this phase of the project, we found out that our first idea, which was manipulating and changing data within the backup file, did not work as we had intended. However, we now know precisely why and how it works. The actual update of the backup file is doable using the injection approach. However, if we update the backup, a new module identity is required for the EdgeAgent to start the updated module. This is done through the Azure IoT Identity Service, which requires an internet connection while creating new identities. For the IoT identity service to create a new identity, it needs to validate its sas token against the token stored in the cloud. This is done by the IoT Identity service over the Internet as it connects to the IoT Hub. However, while we investigated this concept and found out how to communicate with the underlying Edgelet daemon services, we came up with a subconcept based on concept 1. Instead of manually changing the module through decrypting, changing the backup, and encrypting it again, we tried to change modules through the Edgelet API. Unfortunately, this concept did not work either, but because of different reasons, as we bypassed the EdgeAgent complete, we missed out on crucial steps. While updating a module, EdgeAgent does a lot more than starting another docker. It applies several IoT Edge-specific variables and files to the docker container. Because we bypassed the EdgeAgent, we missed out on those which caused the container to crash. We tried appending those manually but did not get the EdgeAgent to orchestrate our modules.

Following our failed attempts to extend the functionality of the Azure IoT Edge and use the internal functionality, we decided to move on to concept 2. In order to prove that the concept is doable and works, we created a prototype that could act as a proof of concept; this prototype was built on the concepts we learned from Azure IoT Edge and the requirement for synchronization and orchestration presented in the previous chapter. It can operate entirely offline and synchronize against a local configuration/digital twin. Based on its digital twin, it orchestrates the different modules specified in the configuration. In the following chapter, we will elaborate on the potential for future development and research in order for Tetra Pak to provide a solution for continuous offline orchestration and configuration.

Chapter 6
Discussion & Related work

In this chapter, we will have a thorough discussion and evaluation of our research and results. Firstly, we will reflect upon our own work. This reflection will highlight the strengths and weaknesses of the study while acknowledging the limitations and threats. This aims to increase our credibility and reliability while also guiding future research. We will discuss the generalizability of our results and to what extent our findings can be applied to other contexts and environments. This section is vital in order to understand the broader view and relevance of our study. Moreover, this chapter will discuss related work focusing on four different papers that we have read throughout this research. We will compare our study and findings within the current academic landscape and discuss how our study aligns, diverges, and advances current research. Lastly, we will discuss the potential for future research and the directions for this.

6.1 Reflection on our own work

In this section, we will reflect on and discuss how our work was performed. We will discuss how the methodology of this paper affected the results and what we could have done differently. We will go into detail about what we have learned carrying out research. The purpose of this is multifaceted and involves many aspects like methodological evaluation, discussing lessons learned, and transparency.

Our thesis work was divided into three parts as we wanted to answer three partial research questions related to a main research question. Thus, the different parts need to be discussed separately. In the first part, we explored the current setup and infrastructure of Tetra Paks IoT systems and, in more detail, the present and in-use Microsoft IoT Edge system. We needed to explore these systems to understand better how they work in the context of Tetra Pak. We explored these systems extensively through reading documentation and source code, but also through testing and meetings with developers. We could have extended this research by including meetings with service personnel as they are using these systems daily.

The second part of our thesis was related to exploring and defining a set of requirements for a system that will allow for digital twin synchronization. We aimed to determine these requirements with the help of the literature and the knowledge and lessons learned from exploring the current Tetra Pak solution. The literature necessary to define how information is synchronized in the best way took time to find and was sometimes hard to interpret. Thus, we partially succeeded in exploring the most current research and were not particularly happy with the results and conclusions. However, we stand by the results as they were crucial for the further definitions of the requirements and later when determining if the current systems were extensible.

The methodology for defining the requirements in this part of our work was mainly an analysis of the current systems and of different research about information synchronization. This method fits the thesis very well, and as we explored the research and the requirements iteratively, this also fits the iterative nature of exploration and investigation. However, we believe that the method could be improved with additional exploration through more real-world testing or from real-world observation of the workflows. These steps could help narrow the requirements and allow for more specific problems to be considered.

The last part of our research was to determine if extending the current IoT systems at Tetra Pak was possible. This was done through prototyping and investigating the programs, documentation, and source code. The only way it could have been done differently was if we had more direct contact with the IoT Edge software creators; thus, we could have directly asked them these questions. However, these people were not available for interviews. Moreover, there is no guarantee that these people would provide the correct answers that we needed or that they would just try to sell us another Microsoft product. Thus, we could not have gotten the answers to whether the current system could be extended any other way than prototyping and investigating the programs.

We could have found other companies or individuals using the same Microsoft systems and seen if they had encountered the same problems. It could have been interesting to see if they had found a solution to the same problem. However, we do not know where we would find this information or if Microsoft would share this information.

6.2 Threats to Validity

In academic and professional research about technology and software, it is very important to recognize and address the possible threats to validity. In order to ensure the credibility of the study, it is necessary to discuss these aspects. The validity part refers to the extent to which results reflect and answer the questions they are intended to. We need to explore various factors that can undermine the validity of our results and findings. By discussing these threats, such as oversight, bias, and misinterpretation, we can enhance the transparency and credibility of this research and provide valuable insights for future research.

Firstly, we are dependent on secondary sources; we rely a lot on Microsoft's documentation. While this provides a good foundation, we may have misinterpreted it and may not represent the complete picture. This can lead to oversight and misinterpretation.

To validate our observations regarding the absence of offline functionalities in IoT Edge, we have reached out to Microsoft and their product team for IoT Edge. However, they can be heavily biased as their perspectives and interpretations are influenced by their own interests

and business perspectives. Moreover, they know how everything is intended to work, which can cause them to not think outside the box.

Our requirements specification is derived from our own research and meetings with technicians, developers, and product owners. These interactions are crucial and very important but limited to the understanding of these individuals. Their experience is based on what they have learned from the online system; it is possible that a few new requirements can be explored when we release our first version and test it in a real scenario. Therefore, our requirements might change as they are iterated and tested further.

The conclusion we have drawn about how to solve this problem is based on our relatively minimal proof of concept, see figure 5.4, and our analysis of the open-source code from the IoT Edge. However, as the IoT Edge codebase is very large, we have not been able to explore everything in detail. This suggests a slight possibility that we might have missed one or more critical components and mechanisms that are key to the orchestrator's functionality.

Lastly, the field of IoT and edge computing is evolving rapidly. Our study is based on the constraints of the current environment and technology. Microsoft has a product on its way that, according to them, will change the entire architecture of their IoT offering. We have presented our research to them, and they understand the problem. A solution to Tetra Pak's problem might be part of a future software release. As our research is heavily focused on the Azure IoT Edge, all the results of this study will not be applicable to other software solutions.

6.3 Generalization of results

Examining the generalizability of the results presented in this paper is crucial in understanding their broader application. By examining the outcomes and discussions, we aim to assess whether they are specific to our paper's context or if they are relevant across different scenarios. This exploration of generalizing our findings is vital to increase the confidence and credibility of our results. Moreover, it is also essential to describe if our solutions offer valuable insights and solutions to others facing similar or related issues.

We will first examine if the conclusions and results of the investigation into the IoT Edge systems and Tetra Paks systems are useful in a more general context. Then, the requirements will be explored, and the examination of the IoT Edge system and the prototyping of our solution will be discussed.

Throughout the exploration and investigation into the systems utilized by Tetra Pak, we discovered how the current solution works in an online environment where the physical devices are consistent with their respective digital twin. We discover how Tetra Pak uses their own tool, the IoT Management tool, which creates updates that are later sent to the Microsoft IoT Edge system to perform these updates on the physical devices. Thus, the exploration and investigation of the IoT management tool might not be applicable in a more general scenario as this system is locked and only available for Tetra Pak and its service personnel. However, the concepts the tool utilizes are comparably the same as the Microsft IoT Edge system supports them. Thus, the results of the exploration into how these tools work in an online scenario are also applicable in a general sense.

Moreover, we explored how the IoT Edge system performs and functions in an offline scenario. The functionality and concepts described should be the same for everyone who uses the IoT Edge system and wants to update an IoT Edge device while offline.

By exploring the IoT Edge system, we discover a set of requirements described in the context of Tetra Pak as requirements. These requirements were also supported by a shorter literature study, which included more general concepts and ideas for what is necessary for a system in order to support information synchronization. These requirements could be very easily generalized as they should be helpful as a framework for anyone who needs to create or define a system that supports Docker orchestration, both in online and offline environments. These requirements will also provide a stable foundation as they are heavily inspired by the functionality of the IoT Edge system, which itself is a stable and secure platform. However, the IoT Edge system does not support offline synchronization and, according to Microsoft, never will. The set of requirements we defined might be the best option for anyone who desires offline functionality at the time of writing this paper. This is because the defined requirements are constructed from an extensive study of an existing and well-functioning program within a particular company's context and a thorough literature study of information synchronization.

Therefore, the results of the last part of this paper, which explores the IoT Edge system in detail, are very generalizable. Research on this system can help anyone wanting to understand the detailed background of how this system works and also for anyone who desires inspiration for a comparable system. Furthermore, the prototyping and creation of our own orchestrator can be generalized as a solution for any type of Docker orchestration. This system correctly manages and synchronizes the system according to a configuration file and incorporates a digital twin infrastructure, further allowing for interoperability between offline devices and online cloud repositories. The prototype can also be very generalized as a solution for docker orchestration and as it can be run and configured to work with every Docker container in any scenario.

6.4 Related work

In this section, we will examine research that aligns with the challenges in our problem statement or the solutions proposed in this thesis. We were not able to find any research regarding our specific context and problem. As explained in the introduction there exists a gap in the research of IoT in an offline environment. However, there is related research on the broader concept of configuration management. The initial focus will be on two articles connected to the core issues regarding configuration management. Subsequently, we will delve deeper into two other articles chosen for their direct relevance to our suggested solutions or the concepts of our results.

6.4.1 Critical Success Factors for Configuration Management Implementation [1]

Summary

The paper "Critical Success Factors for Configuration Management Implementation" by Usman Ali and Callum Kidd has looked into the aerospace and defense industry configuration management (CM). Their research involves finding the critical factors to successful CM processes, which are most important in these high-stakes sectors. This paper's primary contribution is identifying and categorizing 21 critical success factors. These are organized into seven different categories. These categories are meant to give people who work with CM an understanding and act as a guide for enhancing their methods and practices.

The methodology of this research involved both qualitative and quantitative techniques. Firstly, they conducted interviews that went in-depth into different insights about CM. Following that, they did questionnaires to quantify these insights. They concluded that it is imperative how the practitioners implement CM; the practitioner's experience and training greatly influence how they approach their CM. To conclude, this paper guides those practicing CM and those who aim to improve their CM practices' effectiveness and efficiency.

Discussion

Both studies are made in high-stakes environments, meaning there is no room for errors. The study above is made in the defense and aerospace industries, and our studies are in the packaging and food processing industries. This emphasizes the importance of configuration management across different sectors. Both studies acknowledge the challenges that are involved in implementing configuration management systems. Many of the paper's critical success factors are relevant to our requirements, specifications, and prototypes and align with what we have found necessary.

There are a lot of deviations, though; as discussed above, the paper by Ali and Kidd focuses on the aerospace and defense industry and on general success factors without diving into specific solutions available. In comparison, we focus on Azure IoT Edge and the packaging industry. Moreover, our master's thesis explores configuration management in offline scenarios.

Our work contributes to understanding and managing configuration management in an offline IoT environment. And the challenges involved when utilizing IoT solutions inside an intermittent connection scenario.

Both this master's thesis and Ali and Kidd contribute to the field of configuration management but do so in different ways from different perspectives. We are focusing on connectivity and cloud solutions in the packaging industry, while their paper focuses more generally on success factors based on their findings from the defense and aerospace industry.

6.4.2 Clone-aware Configuration Management [17]

Summary

This paper introduces "Clever," a software configuration management system that primarily focuses on clone-aware functionality. In addition to the conventional features found in configuration management systems, Clever extends its support to clone management. This encompasses clone detection and updating, clone change management, validation of clone consistency, and clone synchronization. Clever aims to address the overarching issue in configuration management related to code synchronization by leveraging the code representation of software clones. These clones can be conceptualized as digital twins of different code versions. A key feature of Clever is the representation of source code and clones as trees in Abstract Syntax Trees (ASTs). This enables the system to measure code similarities using vector characteristics and articulate code changes through tree editing scripts. The significant contribution of this paper lies in introducing the Clever program, which is designed to meet the growing demand for effective code clone management.

The paper conducted a comprehensive review and empirical evaluation of the Clever program. Various experiments were performed, each focusing on a specific functionality of Clever. The study's findings indicate that, when applied to large-scale open-source systems, the Clever system demonstrated high efficiency and accuracy in clone detection and updating. Additionally, Clever proved to be effective in providing consistent validation and synchronization of clone changes.

Discussion

This paper is relevant to our work due to the nature of clone synchronization. The paper explores the overall concept of clone synchronization, which in itself can be seen as a parallel to our problem regarding the synchronization of twin devices. However, the suggested paper is more directly involved in the configuration of management within software development and is not configuration management on a higher level of abstraction, such as in our case. Nevertheless, inspiration and application of how these clone changes are detected and supported by Clever can be helpful as a guide for further exploration into how configurations are synchronized between the physical and digital twins.

Our paper and "Clone-aware Configuration Management" enhance the configuration management research in the context of information synchronization. However, the papers differ in the type of data represented and used as a configuration item. Our work further improves their concepts of synchronization of software clones as our research focuses on the more general concept of synchronization of device twins/configurations.

6.4.3 Overview of Docker Container Orchestration Tools [14]

Summary

This paper analyses different Docker container orchestrations in order to help decision-makers make better decisions in the rapidly changing cloud computing domain. In today's landscape, containerization is an essential part of modern development and configuration management. This paper dives into three different tools for the orchestration of Docker containers. The purpose is to aid practitioners in understanding the differences and similarities in order to make informed decisions. This approach not only highlights the theoretical aspects but also provides details of the limitations and practical applications of these tools. They do this by employing a combination of performance metrics, technical analysis, and real-world studies and examples. Their conclusion is that all tools have their advantages, and the choice is highly dependent on the specific deployment scenario. These differences include scalability, community support, and how easy the tool is to use. Their conclusion emphasizes the need for different solutions tailored to the specific environment and technology.

Discussion

Like our studies, they emphasize the need for tools in configuration management. Our study discusses Azure IoT Edge and our own orchestrator concept, both of which have the functionality to orchestrate Docker containers. This paper compares and discusses different Docker orchestrators. This paper provides a technical analysis of those orchestrators while we examine the mechanism and challenges of using IoT Edge inside a specific scenario. Both studies aim to help practitioners make informed decisions, although their paper is more general.

What differs in this study is their specific technological focus. The paper focuses on Docker orchestration, whereas our research focuses on Azure ioT Edge in Tetra Pak's operations and context with a strong focus on offline capabilities. Our thesis is unique in that it addresses the challenge of offline configuration management in an IoT environment.

Our research contributes to the work of this paper as we have explored another Docker orchestration tool, Azure IoT Edge. Both works are relevant for practitioners and can offer guidance in implementing and selecting software for configuration management in different contexts and environments.

6.4.4 A Dynamic Hierarchical Framework for IoT-assisted Metaverse Synchronization [6]

Summary

The metaverse is a comprehensive rework of the workplace environment we currently have. This paper addresses some of the more challenging synchronization aspects in the metaverse, focusing on IoT devices. The authors have identified a need to adapt current solutions to the changing landscape of IoT and virtual environments to more efficient and dynamic frameworks.

This paper's main contribution is introducing a hierarchical framework that dynamically synchronizes IoT devices and their digital twins in the metaverse. The framework is designed to be adaptable and scalable, improving compared to the existing static models. The methodology for this paper was the creation and evaluation of the hierarchical model that categorizes the IoT device's interaction with the metaverse. They used a combination of simulation and real-world testing to validate their creation, focusing on efficiency, adaptability, and scalability. The paper concludes that the framework significantly enhances the interaction between IoT devices and the metaverse. Moreover, the authors want to highlight the importance of dynamic synchronization in future technologies and landscapes.

Discussion

This paper is about the overall solution of data synchronization between devices and their digital twin, which is highly applicable to our problem statement and solution. They developed a framework to utilize IoT devices to sense and record information to enhance the synchronization between physical devices and their digital twin representation. However, this framework considers the digital twins to have a greater responsibility as they are actual simulations of real-world objects. Nevertheless, the framework is similar in the desired functionality of digital twin synchronization, even if the devices mentioned in the paper are always online. The paper's conclusions offer valuable insights and perspectives for enhancing digital twin synchronization. Both papers enhance the research into configuration management, and in particular, they both further the knowledge of the synchronization of physical devices and their device twins through the utilization of IoT devices.

6.5 Future Work

Through our research, we have encountered limitations and unexplored areas. These point us toward continued research and provide opportunities to further understand how IoT solutions can operate under intermittent connection scenarios. In this section, we will present several key areas that can be researched further. These suggestions aim to guide future research to build upon our work and explore relevant aspects in this field of study.

One of the most important directions for future research involves researching additional contexts and environments. Our study is very focused on Tetra Pak's context and environment, but it opens up possibilities for exploring similar challenges in different contexts and environments. Future research can be done on others who are using IoT solutions in intermittent connection scenarios. This could be done in order to test the generalizability of our study and solution and to research if other companies are experiencing similar issues. This expansion could broaden the result and enrich understanding.

Future research should also focus on refining and extending our methodology. By collecting data from other types of real-world scenarios, we could have gathered statistics and nuanced insights. Moreover, future research can expand upon our analysis of Azure's IoT Edge and analyze more tools in order to evaluate if they have addressed these issues or solved them in other ways.

The field of IoT and edge computing is moving forward swiftly and evolving every day. As new technologies and products reach the market, the relevance and applicability of our findings may be impacted. Future studies should aim to understand and analyze the implications posed by these changes and focus on researching how offline scenarios can be handled in the context of future products and IoT solutions.

Lastly, future research should explore the questions raised by our findings and aim to build upon our research by expanding on these areas. It can dive into aspects that were beyond our timeframe and scope and investigate those. Addressing this not only extends our work but also helps drive the continuous improvement and integration of IoT Systems and knowledge in the field.

Chapter 7
Conclusion

Based on the research we have done, it is not possible to perform offline updates to an Azure IoT Edge device while it is disconnected from the internet.

Every change to a module requires a new identity. Identities come from the IoT Identity service, which is a service developed by Microsoft to ensure secure IoT services.

Old module IDs can be used offline, but new ones cannot be created and authenticated. This is because the identity service uses SAS authentication against the cloud. When a new module is requested, there are several authentication steps involved. In the last step, the IoT Edge device sends its SAS token to the cloud, and the cloud(IoT Hub) checks the tokens against the SAS key registered when the Identity for the IoT Runtime was created. If there is not any connection present at the time it requests for a new identity, the request will not go through.

We examined the core mechanism of the IoT Edge system and other open-source runtimes. Then, we did a literature study on information synchronization and version control in the context of configuration management. Based on the study and the IoT Edge system, we developed a set of requirements for a software system to solve the problem of offline updates to IoT devices. These requirements were based on Tetra Pak's context and their specific problems but also the general limitations of the current system.

The essential requirements revolved around the stability, security, and compatibility needed in a software system running under the challenging conditions of Tetra Paks sites worldwide. The solution needs to run production critical applications responsible for food safety and the operation of dangerous machines.

The solution must be fully compatible with Docker, as all software running on the current solution is containerized and executed through Docker. The solution must be able to manage and orchestrate the docker containers. Tasks like restarts, updates, pulls, builds, etc must be handled seamlessly.

Furthermore, the runtime must be secure and protected from unauthorized access. Changes should be made only through the IoT Management tool. By implementing this, Tetra Pak can guarantee that only people with access to their IoT Management tool can make changes. Fur-

thermore, as all changes go through the IoT Management tool, we can keep its database as the single source of truth, making the solution compatible with the current synchronization concept.

References

[1] Usman Ali and Callum Kidd. Critical success factors for configuration management implementation. *Industrial Management & Data Systems*, 113(2):250 – 264, 2013.

[2] Wayne A Babich. *Software configuration management: coordination for team productivity*. Addison-Wesley Longman Publishing Co., Inc., 1986.

[3] Docker Inc. Docker Docs. https://docs.docker.com/.

[4] Peter H Feiler. *Configuration management models in commercial environments*. Carnegie Mellon University, Software Engineering Institute Pittsburgh, PA, 1991.

[5] Gerhard, Rieger and FSF and Linux developers community and Open Group. socat. http://www.dest-unreach.org/socat/doc/socat.html.

[6] Yue Han, Dusit Niyato, Cyril Leung, Dong In Kim, Kun Zhu, Shaohan Feng, Sherman Xuemin Shen, and Chunyan Miao. A dynamic hierarchical framework for iot-assisted metaverse synchronization. 2022.

[7] Microsoft. Azure IoT Identity Service. https://azure.github.io/iot-identity-service/.

[8] Microsoft. Edge Agent. https://github.com/Azure/iotedge/tree/main/edge-agent.

[9] Microsoft. Edge Hub. https://github.com/Azure/iotedge/tree/main/edge-hub.

[10] Microsoft. IoT Edge Security Daemon. https://github.com/Azure/iotedge/tree/main/edgelet.

[11] Microsoft. Properties of the IoT Edge agent and IoT Edge hub module twins. https://learn.microsoft.com/en-us/azure/iot-edge/module-edgeagent-edgehub?view=iotedge-1.4.

[12] Microsoft. Understand Azure IoT Edge modules. https://learn.microsoft.com/en-us/azure/iot-edge/iot-edge-modules?view=iotedge-1.4.

[13] Microsoft. Understand the Azure IoT Edge runtime and its architecture. https://learn.microsoft.com/en-us/azure/iot-edge/iot-edge-runtime?view=iotedge-1.4.

[14] Marek Moravcik and Martin Kontsek. Overview of docker container orchestration tools. *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA), Emerging eLearning Technologies and Applications (ICETA), 2020 18th International Conference o*, pages 475 – 480, 2020.

[15] Rust Foundation. Rust a language empowering everyone to build reliable and efficient software. https://www.rust-lang.org/.

[16] Vinay Singh, Amarjeet Singh, Alok Aggarwal, and Shalini Aggarwal. Advantages of using containerization approach for advanced version control system. In *2022 Fourth International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT)*, pages 1–4. IEEE, 2022.

[17] Nguyen Tung Thanh, Nguyen Hoan Anh, N.H. Pham, J.M. Al-Kofahi, and T.N. Nguyen. Clone-aware configuration management. *2009 IEEE/ACM International Conference on Automated Software Engineering, Automated Software Engineering, 2009. ASE '09. 24th IEEE/ACM International Conference on*, pages 123 – 134, 2009.

Appendices

EXAMENSARBETE Adaptive Synchronization and Orchestration: Tackling Offline and Intermittent Connectivity in IoT Environments
STUDENTER Gustav Engström, Victor Gunnarsson
HANDLEDARE Lars Bendix (LTH)
EXAMINATOR Emelie Engström (LTH)

Adaptiv Synkronisering och Orkestrering av IoT enheter med dålig uppkoppling

POPULÄRVETENSKAPLIG SAMMANFATTNING **Gustav Engström, Victor Gunnarsson**

I takt med att mjukvara blir mer och mer centralt inom industrin ökar kraven på kontinuerliga mjukvaruuppdateringar, samtidigt driver AI-trenden i världen vikten av datainsamling och uppkoppling mot molnet. Mot bakgrund av detta har Tetra Pak börjat koppla upp sina maskiner med IoT lösningar, ett steg som resulterat i flertalet nya utmaningar.

Tetra Pak har utvecklat en molntjänst som använder sig av IoT teknik för att hantera konfiguration och datainsamling av deras maskiner centralt från Lund. Lösningen bygger på Microsofts Azure IoT Edge för att hantera uppkoppling, orkestrering och datainsamling.

Genom denna IoT Edge kopplas maskiner och fabriker upp som "edge" enheter mot molnet för att möjligöra fjärrstyrning över internet. Varje "edge" enhet har en digtal tvilling i molnet. När tvillingen uppdateras synkroniserar edge-enheten sig automatiskt. Detta system är ett skalbart sätt att genomföra snabba och effektiva uppdateringar.

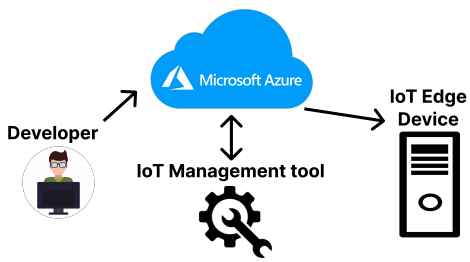

Figure 1: Software distribution overview

Många av Tetra Paks fabriker ligger ute på landsbyggden vilket medför stora problem med uppkopplingen. I vårt exjobb har vi tittat på hur man på ett säkert sätt ska kunna uppdatera mjukvaran på maskiner utan tillgång till internet. Lösningen måste uppfylla tre huvudkriterier: För det första måste lösningen garantera säkehet i hanteringen av konfigurationer. För det andra, måste lösningen integreras med befintliga system och vara kompatibel med den nuvarande arkitektur. Slutligen, måste systemet vara stabilt, snabbt, resurseffektivt och kunna köra i samma miljö som nuvarande IoT Edge-enheter.

Efter en analys av den befintliga IoT Edge lösningen kunde vi konstatera att det inte finns möjlighet att stödja offline uppdateringar via denna. Vidare har vi fastställt att Microsoft inte har några avsikter att stödja detta i framtiden heller. Mot bakgrund av detta tog vi fram en kravspecifikation för en lösning över hur "edge" devices kan uppdateras utan internetuppkoppling. Med hjälp av vår kravspecifikation utveklade vi därefter en applikation som demonstrerade den praktiska tillämplingen av vår föreslagna lösning.