FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

CPS/IoT Ecosystem:

Exploring operational scopes for industrial Internet-of-Things and an analysis of Quality-of-Service properties

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Ádám Dukkon
Matrikelnummer 11703504

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Mitwirkung: Projektass. Dipl.-Ing. Haris Isakovic, BSc

Wien, 20. August 2019

Ádám Dukkon	Radu Grosu

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

CPS/IoT Ecosystem:

Exploring operational scopes for industrial Internet-of-Things and an analysis of Quality-of-Service properties

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Ádám Dukkon
Registration Number 11703504

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Assistance: Projektass. Dipl.-Ing. Haris Isakovic, BSc

Vienna, 20th August, 2019

Ádám Dukkon	Radu Grosu

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Ádám Dukkon
Balfi út 71, 9400 Sopron Hungary

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. August 2019

Ádám Dukkon

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

This thesis addresses the problem of monitoring the Quality of Service in CPS/IoT systems. In such heterogeneous systems, where each component is from different vendors and use diverse programming languages, the configuration, deployment, and maintenance is challenging. In order to overcome this problem, performed a QoS analysis with the result of a quality model. This quality model is used to compare different IoT use-cases based on their QoS requirements. As an IoT architecture, we suggest a scalable, fault-tolerant solution, based on Arrowhead and Docker swarm, with a deployment workflow and monitoring system. To ensure a certain QoS level within the system, we implemented a QoS monitoring and reconfiguration service. This service ensures that the system works optimal, it detects and reacts to the possible QoS violations. As an evaluation, we provide a series of experiments, and an industrial use-case, which validates our solution in a real-world scenario.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Abstract **vii**

Contents **ix**

1 Introduction **1**
1.1 Motivation and Problem statement 1
1.2 Structure of the work 7

2 Theoretical background **9**
2.1 Internet of Things 9
2.2 Cyber Physical Systems 16
2.3 Service Oriented Architecture 18
2.4 Arrowhead Framework 20
2.5 Containerization 23
2.6 Continuous integration and deployment 26
2.7 Runtime Monitoring 28
2.8 Quality of Service analysis of a CPS/IoT system 39

3 State of the art **51**
3.1 Related works 51
3.2 Comparison and summary 54

4 Methodology **55**

5 Implementation **59**
5.1 Hardware infrastructure 59
5.2 Docker implementation 61
5.3 Deployment workflow 64
5.4 Monitoring architecture 66
5.5 QoS reconfiguration service 67

6 Experiments **71**
6.1 Service start leaking memory 71
6.2 Service with increased CPU usage 73

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

6.3 Rollback updated service 75
6.4 Migrate the service to another node 76
6.5 Leaking service in multiple services environment 77
6.6 Different programming languages 79

7 Industrial use-case **83**
7.1 Simulation 83
7.2 Data collection 88
7.3 Data Storage 89
7.4 Data visualization, and Simulation control 94

8 Future work **97**

9 Conclusion **99**

List of Figures **101**

List of Tables **103**

List of Algorithms **105**

Acronyms **107**

Bibliography **109**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

1.1 Motivation and Problem statement

The Internet of Things (IoT) concept was composed in 1999, by the Radio Frequency Identification (RFID) development community, and its relevance is constantly increasing [1]. Internet-of-things(IoT) is a network of physical objects, which enables them to exchange data with each other [2]. These physical objects - *things*- can be nearly anything: smartphones, sensors, actuators. With Cyber Physical System (CPS) we can combine the physical world - the *things* - with computing capabilities from cyber world [3]. We use CPS with the IoT to collect data from the physical environment. Then we analyse this data and based on the results we can control the system through software services [4]. Devices and services are heterogeneous in a cyber-physical system, therefore we need an architecture, which is capable of integrating them together. For such situations, a Service Oriented Architecture (SOA) can be used, since it enables the interoperability among services from different sources [5]. One SOA architecture, for example, is the Arrowhead Framework [6]. We observe IoT systems as hierarchically composed infrastructure with three scopes of operation, as depicted in Figure 1.1 [7].

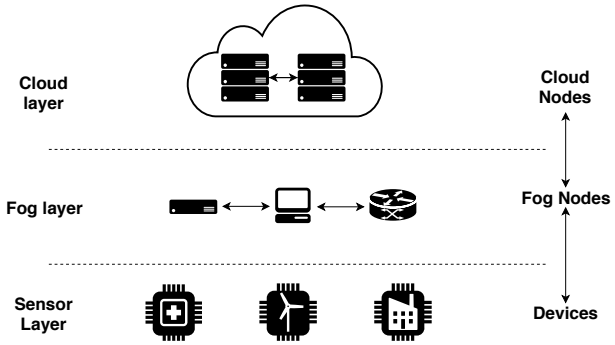


Figure 1.1: Basic architecture of a CPS/IoT system

With the integration of IoT and CPS, we can build an architecture which helps us achieve higher productivity in a production plant. In a factory, we can integrate the existing infrastructure, such as production robots, conveyor belts, production lines, with a CPS/IoT system. With this technology, we can extract data from these systems, and analyze the gathered data for possible improvements [8]. This is one of the key features for the transformation in the industrial world, called Industry 4.0. Industry 4.0 refers to the fourth industrial revolution in the industrial sector. In order to understand its significance, we collected the main changes from the previous three revolutions:

1. The first industrial revolution was about the transformation from hand production to machines (mechanization), and an increasing spread of the steam machine
2. The second industrial revolution was about the usage of the electricity in order to make mass production
3. The third industrial revolution was about the usage of robotics and electronics, and information technology

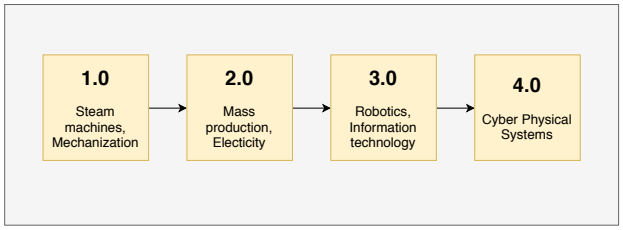


Figure 1.2: Industrial revolutions timeline.

The term Industry 4.0 (or Industrie 4.0 in German) was originally promoted by the German government [9], more precisely by the Federal Ministry of Education and Research for a project of the high-tech strategy in 2011.
In the following, we collected some manufacturing changes, which are force the industrial evolution [10]:

- First is the so-called time to the market period: this refers to the amount of time from the initial idea until the product is available in the market. With a faster development time, we can shorten this period.
- Second is the increasing individualization. Any single user would like to have the option to customize their product. This trend is also known as batch size one production. This kind of production claims highly flexible solutions from the development and the manufacturing side as well, in order to serve all different versions.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- Third is the increasing value of the resources and raw materials. This environmental modification forces the manufacturers to adjust their processes in a more resource efficient way.

The previously described IoT and CPS techniques have a key role in the fourth industrial change [11]. The IoT ready robots in the factories have the ability to do real-time data collection, and analysis via a CPS system. Then we can use this data in several applications [12] [13]. As an instance, predictive maintenance is one of these applications. With a predictive maintenance algorithm, we want to predict possible machine failures beforehand. Predictive maintenance can save lot of money and production time, since we can minimize unplanned downtimes and failures. As a second use-case, we can examine the performance tracking via Overall Equipment Effectiveness (OEE). OEE is a standard measurement for manufacturing productivity: it is calculated by the multiplication of availability, performance, and quality. OEE contains three different measure value, therefore it is not enough to be good at one: if the line produces fast, but with the bad quality, the overall OEE will be low. Since there is data about everything in the CPS system, the OEE can be calculated and tracked real time, to intervene if one of the measurements is critically low. Another considerable usage area of the CPS/IoT systems is the creation of more advanced simulations of the industrial assets. With the simulations, we can model a digital twin from the factory, which is identical to the real world. It has the advantage, that we can simulate new robots, or try multiple configurations before they are used in the real environment. This saves much cost and time since there is no need to interrupt the production process.

Although there is a lot of potential in these systems, we have to recognize the possible risks, and challenges as well. These systems operate constantly in three shifts, in a highly regulated way with strict rules and policies [14]. It is necessary to have these rules since any failure in the system can result in injury or economic loss. In order to meet the industrial demand, we need to provide a highly dependable system, which can assist the production continuity. “The dependability of a system reflects the user’s degree of trust in that system.” [15]. Dependability has some principal properties: security, safety, availability, and scalability. Security features of the system ensure that unauthorized users cannot have access for the operation within the system. With safety properties, we want to guarantee, that the service can operate without making damage or injury to its environment. Availability for example in case of a conveyor means, that it operates 99 per cent of the year without interruption. In a factory, the installation of a new conveyor in the production line is also a possible scenario, the control unit has to deal with it, which refers to the scalability characteristic of the system.
Dependability properties are parts of an overall service quality measurement, namely Quality of Service (QOS). ITU-T Rec. E. 800 defines QoS as "collective effect of service performance which determines the degree of satisfaction of a user of the service" [16]. The system performance depends on diverse quality factors and properties, which form a quality model together. Although, there are existing quality models, we need to create our own quality model due to the CPS/IoT system special characteristics.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

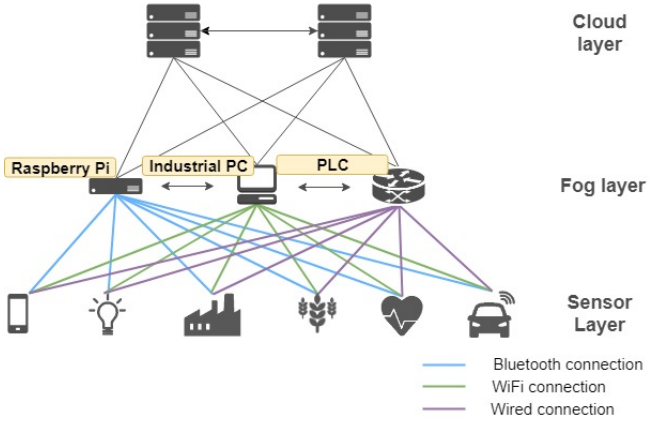


Figure 1.3: CPS/IoT architecture heterogen hardwares

As it can be seen in the Figure 1.3, CPS/IoT is a complex system with special characteristics. The system consists of a high number of heterogeneous components: distributed sensors from different vendors in the sensor layer and various physical elements, such as industrial PC, Raspberry Pi or PLC in the fog layer. These machines have a different degree of computational, communicational capabilities. Scalability is also an important characteristics since the system needs to handle a growing amount of sensors.
In order to encompass these devices into a CPS/IoT system, we need to provide the interoperability among them. Interoperability means that they can connect, and communicate with each other via pre-defined interfaces. This is challenging due to the mentioned various communication abilities, and the numerous possible communication protocols. For instance, 6LowPAN [24] protocol supports low-power mode: as a comparison, 6LowPAN [24] consumes approximately 106mA, while WiFi [25] 300mA [26]. This is an important difference for the sensors with limited power supply. However, in another use-case such as industrial applications, reliability and security are more important, therefore WiFi is a more suitable choice for those applications.
The management of diverse machines is also challenging, as they serve different roles, with various resource capabilities, different CPU architecture or operating system. For these reasons, we can not treat them equally, which causes problems during service deployment, and system monitoring.

In a traditional software development process, the developers implement the services, then build an executable application from the source code. After building the application, they ship this executable service into a suitable node of the system. This transfer is often done manually. As a last step of the deployment process, the developers need to start or update the service. As it can be seen, this process includes several manual operations, which are a possible sources of errors. In addition to that they make the process slow. On the contrary to these manual process, these steps can be automated with a deployment system. This automation makes the shipment of the services and fixing the software failures more quickly.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

As it was previously described, we have to consider that the nodes in the system have different abilities, which is possibly, not suitable for the service. This leads to several difficulties in the process of deployment, such as:

- a service might have a greater resource requirement than there is available in a certain node
- two services cannot be deployed together, because one is delay sensitive, while the other one is resource intensive and they would interfere with each other
- a service can operate only on a specific architecture or operating system
- one node has a wired connection to the robot, and we want to force the service to run exclusively on that node.

During the deployment process, we can specify multiple placement constraints, regarding the hardware, software, or software interaction. Since there are multiple nodes within the architecture and usually more than one service can be deployed into the same node, along with the placement constraints, we end up with many combinations of the possible service deployment scenarios. This makes the manual deployment impossible in the CPS/IoT architecture. As we can see deployment is a challenging process, Arrowhead uses manual build and deployment methods at the moment, and this work is part of an a ongoing activity to ensure fully automated CI/CD.

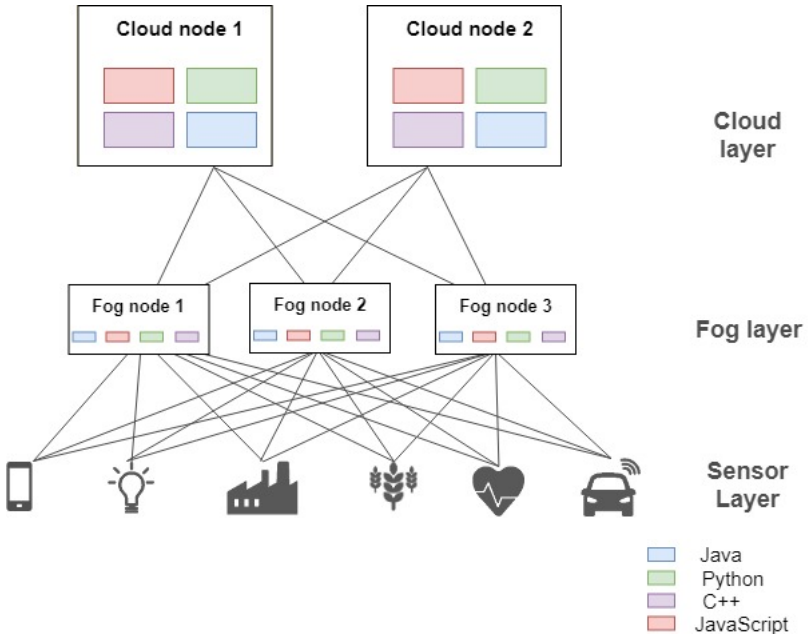


Figure 1.4: CPS/IoT architecture heterogen softwares

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

There is another problem about the deployment, which comes from the SOA architecture. In SOA there is no restriction regarding the used programming language of service. This means we have to deal with a wide variety of programming languages. In SOA every service is independent, developed by a separate developer team and use diverse programming languages. The developers can use the architecture's native language (Java in Arrowhead Framework) or any other language, which can be integrated into the system, as it can be seen in the Figure 1.4. Usually the languages have multiple versions and the services have multiple dependencies: for instance, an average Arrowhead core service has 20 dependencies. The problem occurs when we have to install all the dependencies and language versions manually to a large scale environment. Manual installation is not only time-consuming but it can cause a lot of problems, for instance if we install the wrong version from the language. Another problem can be if we do not install a necessary dependency into a node before the deployment. The challenge about the independent teams is that as developers follow an implementation process, they test their services in a development environment, however, it is possible that the application acts differently in the production environment.
With automatic deployment, we have the ability to encompass the source codes of the services with their dependencies, and runtime environment, thus we do not have to install them manually, which often causes problems.

After the deployment of the services, we want to ensure that they can work together and the system works optimally. In the scope of CPS/IoT and SOA this means that they can interact with each other. In addition to that, all the services have enough computational resource, and none of them interferes with the others. Furthermore, all the functionality of the service is available for usage.
In order to verify the optimal operation within the system, we need a monitoring system. This system collects all the necessary metrics from the distributed nodes in real-time. However, due to the previously mentioned characteristics, the usage of a monitoring system in a CPS/IoT architecture is challenging.
The monitoring solution has to handle all the possible hardwares within the system while it uses a minimum possible computational resource. Since a CPS/IoT system is constantly changing, the monitoring solution needs to deal with that, which means, if we add a new service, or a new node, the system can automatically monitor those as well, without any additional configuration. The real-time data acquisition is also a problematical task, due to the high number of devices, the number of metrics, and the velocity of the data. For this, we have to find a suitable storage solution as well, which can be utilized to store the data.
However, a monitoring system is not enough, we have to specify a model, which describes the optimal behaviour for the system with a formal language. This description then can be interpreted by the monitoring system, which observes the performance of the individual services, and the overall performance per nodes.

A suitable monitoring solution helps to:

- identify the software and hardware failures faster which saves time and money

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- identify possible bottlenecks within the system
- can help analyse failure trends
- it can send a notification to the right people.

From a high dependable system, we expect to operate correctly in presence of any failure. This means that the system needs to be prepared for hardware problems, such as node failure, or the interruption in the communication between two nodes. Failure also includes software problems, such as service interruption. Another complication is, when a service consumes too many computational resources and by this it interferes the other services. A further possibility is that the behavior of the service changes after an update, for instance, it cannot interact and work together with the other services or it even fails to startup. In the traditional process, we have to stop the system and analyze the root causes in such cases. However, a highly dependable system cannot be stopped for a long time, therefore we need a solution. Since monitoring is just a passive observation activity, it can only performs an alert in case of any malfunction. To make the system able to immediately react to this malfunctions, we have to develop an automatic reconfiguration and redeployment service. This service ensures that the system is robust and adaptable against unexpected hardware or software behaviors. To achieve this, the service can take corrective actions in case of any QoS violation, based on its application based knowledge, and predefined problem resolutions.

During the work we intend to engage with the following research questions:

RQ: What are the requirements of establishing a suitable Deployment and Monitoring solution for heterogeneous CPS/IoT system?

RQ: What are the specific QoS properties of an industrial IoT system?

RQ: How to implement a dynamic reconfiguration and redeployment algorithm for service-oriented IIoT systems, given QoS requirements and application-specific hardware and software constraints?

In the scope of this thesis, a container based continuous integration and deployment, along with a monitoring architecture, and a dynamic Monitoring and reconfiguration service will be implemented and configured.
In order to identify use-case specific Quality of Service factors, a Quality of Service analysis will be performed. Based on the result of this analysis, we are going to be able to compare the different CPS/IoT use-cases and classify their quality attributes.
Finally, experiments will be performed in order to test the implemented services. Besides the experiments, we are going to build a demonstration industrial system. This system is going to be used as an environment, where the deployed solutions can be evaluated in the real-world conditions.

1.2 Structure of the work

The thesis is structured as follows:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- **Chapter 2**
  Describes the basic terms and concepts in IoT and CPS field. This chapter also contains the Quality of Service analysis. In the end, there is a comparison between different IoT application use-cases, in order to identify industrial Quality of Service properties

- **Chapter 3**
  In this chapter we describe the current State of the art, and related works regarding to the deployment, runtime monitoring and Quality of Service. We also make a comparison between our proposed solution and the existing Arrowhead implementation.

- **Chapter 4**
  Gives an overview of our methodology for the work. We include the used hardware infrastructure and the used software framework as well.

- **Chapter 5**
  Provides the actual implementation of the suggested solution. This includes the description of implemented algorithms and the used configurations

- **Chapter 5**
  Involves all the performed experiments with the implemented Quality of Service monitoring and reconfiguration service

- **Chapter 6**
  Describes the implemented real-world industrial use-case. For each component we justify our choice of technology and express the reasons, and experiments based on which we have made our decisions

- **Chapter 7 - 8**
  Summarizes the work, and give suggestions for the future work

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Theoretical background

2.1 Internet of Things

According to some studies, the number of IoT devices will reach 100 Billion in 2025, and become a 10 trillion dollars market [17]. Internet-of-things (IoT) is a network of physical objects, which enables them to exchange data with each other [2]. There is considerable diversity in the type of these devices: it can be everything from a PC to a production robot, or a relatively simple temperature sensor. However, they have some similarity: all of them are connected to the Internet and interact with each other via different networking protocols. These devices are often equipped with sensors, which make them suitable to collect data from their environment. The collected data then can be used to describe the physical world. IoT is an emerging technology, and nowadays it is used in many different applications such as Home automation, Smart Agriculture, or - as in this thesis - in the industrial sector. However, there were similar technologies before IoT. A foundation concept for IoT is the RFID technology [18]. RFID enables wireless communication data transformation between microchips. All microchips - called RFID tag - have a unique identifier, therefore it can be used to identify, track or monitor objects. RFID technology widely used is used inter alia in the field of logistics, health care, security, and retailing. Another significant technology before IoT is the Wireless Sensor Networks (WSNs) [19], which is used interconnected sensors to monitoring and sense the surroundings. Most of the WSNs application areas are from environmental and industrial monitoring.

2.1.1 Fog computing

With the emergence of the IoT devices and their generated data, there was a need for an architecture, which is capable to process and store this amount of data. In previous years, this was the cloud computing architecture. In cloud architecture, there is a centralized host, with nearly boundless computational resources, and high bandwidth. The IoT

nodes, which can be a smartphone, or a sensor, or any other device, send their data to the cloud.
However, cloud architecture has two main disadvantages from Quality of Service aspects: there are just a limited number of data-center in the world. This means, the period of time it takes to send the data to the data-center, and receive the signal back, could be more than the tolerated delay in the system. Another issue is about the bandwidth: the speed of data transportation could be limited in the IoT devices, which occurs a bottleneck in the system. For instance, a self-driving car from Google generates nearly 1 Gigabyte of data in every second [20]. There is a possibility that the bandwidth between the car and the data-center cannot handle that much data, which occurs QoS related problems.

After the QoS issues with Cloud computing, there was a need for another architecture. While the data is generated in IoT devices, it would be logical, to process it there. There is an architecture for this, called Edge computing. In Edge computing the processing of the data is done at the edge of the network [21], by the device itself. In this case, the device does not send the data to a remote cloud for processing, therefore we can avoid the bandwidth problem. With this approach we improve the speed of data transportation, so we can support latency-sensitive applications. However, it has another drawback: the computational power of IoT devices, such as CPU, and storage could be limited, as well as the battery life. With the limitation, we also violate the QoS requirements, such as resource utilization.

The third architecture is Fog computing, which is initially proposed by Cisco in 2012 [22]. In fog computing, there is another layer between the edge and the cloud [23] [24]. This fog layer is responsible for the consistent latency, and computational services between the end devices, and the data-centers in the cloud. With the fog architecture, we can process our data closer to the actual device, in order to avoid latency and bandwidth issues. It supports real-time applications, since the computational power is nearby to the device, however it does not suffer from the resource limitation as much as the IoT devices. With fog architecture, we can build a system, where we process the data locally in the fog machines and store the processed data in the global cloud nodes, which is accessible from everywhere.

The Fog computing architecture consist of layers. A layer is an abstraction level, which enables us to encapsulate the working units in a subsystem. We can distinguish three layers:

- **Sensor layer**
  In the sensor layer, there are IoT devices, which are small to medium performance, such as PLCs, IoT gateways, or Industrial PCs. They are capable of operating in real time and interaction with the physical world, but they have limited communication resources and power supply. In this layer, there is no need for powerful machines since resource-intensive algorithms are not running on them.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- **Fog layer**
  Devices related to fog node have more power, typically medium to medium-high performance. These machines still capable of doing real-time communication, however they are equipped with more computational power, and better communication capabilities. The fog layer serves the Extract, Transform, Load (ETL) jobs and higher level control (process control). The main purpose of ETL jobs is to aggregate, filter the data from the lower level.

- **Cloud layer**
  The third is the cloud layer where massive servers operate with multi-purpose machines capable of doing the extremely high-performance computation to handle a massive amount of sensor-generated data. These machines have huge storage and high power consumption. Here data processing, storage, visualization, and analysis are done. The cloud infrastructure could be owned by the company - it is called a private cloud - or hosted by a well-known provider, such as Amazon [25] - called a public cloud. Both of these approaches have advantages – in a public cloud system maintenance of the servers is done by the provider, and the payment is on-demand. In the other hand, many companies do not want to send data outside the company, therefore they implement a private cloud solution.

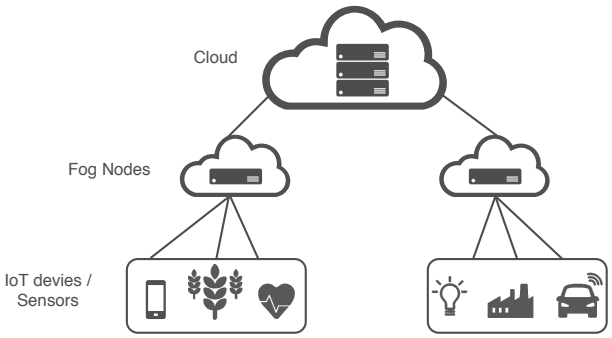


Figure 2.1: Fog architecture

2.1.2 IoT communication protocols

In order to exchange information between the distinct layers in the Fog computing architecture 2.1, we need to employ various communication protocols. Between the Fog and Cloud layer we can use the the ordinary protocols, however they are not entirely suitable for the Sensor layer. There are two main communication protocols, which sufficient for the special requirements of Sensor layer, such as unreliable networks, low-latency and limited power consumption.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

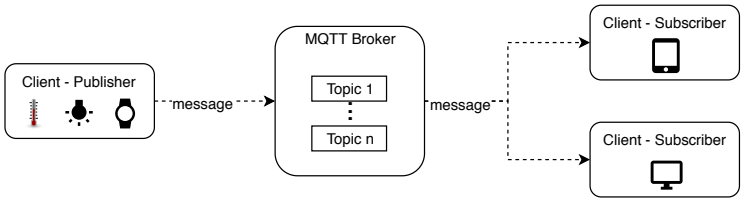
Message Queuing Telemetry Transport (MQTT)

Message Queuing Telemetry Transport is a publish-subscribe type messaging protocol, invented by Dr Andy Stanford-Clark of IBM and Arlen Nipper of Arcom in 1999 [26]. This protocol is extremely simple and lightweight. It was designed especially for small devices such as PLCs which operates with low-bandwidth and poor, unreliable network conditions. There are some main design principles were taken count during the implementation of the MQTT protocol:

- Minimise network bandwidth
- Minimise device resource requirements such as the battery, or computational power (such as CPU)
- Some degree of assurance of delivery. In MQTT there are three levels of delivery assurance, called Quality of Service (QoS):
  - QoS 0 - At most once: best effort delivery. There is no guarantee, that the message will delivered to the receiver.
  - QoS 1 - At least once: the message is delivered at least one time to the receiver.
  - QoS 2 - Exactly once: the message is delivered only once to the receiver.

These are the predefined agreement between the sender and receiver that defines the guarantee of delivery. These design principles make MQTT perfect protocol for scenarios, such as M2M - machine to machine communications, where the bandwidth and battery power are extremely important and expensive resources. As can be seen, this protocol supports QoS in different levels: directly by the three levels of the delivery, indirectly by the resource consumption optimization. The MQTT QoS levels are one example of the Sampling parameters from Perception layer QoS indicator 2.8.3.
MQTT is an ISO standard [27], with many implementations from different kinds of programming languages from C to JavaScript, therefore we can integrate it into various kinds of projects.

Architecture overview:
An MQTT implementation always contains a broker server and as many client as required. A broker server holds the topics. Each clients have to connect the broker and they can publish or subscribe to a topic. MQTT works over Transmission Control Protocol (TCP) protocol which provides reliable, ordered and error-checked delivery of the message payload.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Figure 2.2: MQTT Architecture

Open Platform Communications (OPC)

Open Platform Communication foundation is a set of standards and specifications for the industrial area to real-time exchange data between production machine's control devices - such as PLCs - and client applications [28]. The first release was in 1996 and it only operated on Windows operating system with the main purpose of translating the Human Machine Interface (HMI) or Supervisory Control and Data Acquisition (SCADA) instructions to machine code and the other way round. The original name of the foundation was Object Linking and Embedding, but it was changed to OPC when the process control was implemented. As it was mentioned, OPC has a series of standard specifications:

OPC Data Access (OPC DA) [29] As part of OPC Classic group [30], it allows access to the actual data, such as PLC variables. In OPC DA, we can see only the actual data, we do not have access to the historical data. OPC DA data contains three elements:

- Data value
- Quality information about the data
- Timestamp

OPC Alarms and Events (OPC AE) [31] Also part of the OPC Classic group, it allows exchanging alarms and events type messages. These messages contain three elements:

- alarm and event type message information
- State of the variable
- State of the management

OPC Historical Data Access (OPC HDA) [32] The third member of the OPC Classic group, it allows accessing the historical data from the database. This standard specification defines the OPC server, which stores the data and routines, how clients can retrieve the data.

OPC Unified Architecture (OPC UA) [33]
OPC UA standard specification is originally released in 2008 with the aim of combine all the functionality from OPC Classic. This is a cross-platform implementation developed under the GPL 2.0 license, in Service-oriented architecture (SOA) way. OPC UA is implemented in different programming languages such as Java, C++, .NET, Python. The main focus was in this release on the communication and interaction between the industrial machines and the IoT applications in order to collect the data from the machines and control them through applications.

OPC - similarly to MQTT - has a server-client architecture, where the PLC connects to the OPC server, and the clients can read and write the provided sensor attributes through the server. The server has its own metamodel, how the attributes are actually modelled.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

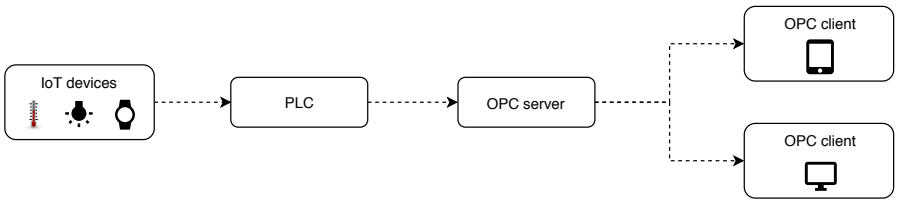


Figure 2.3: OPC Architecture

2.1.3 Extract Transform Load (ETL)

As it was written, the Fog layer is responsible for Extract, Transform, and Load the data from the sensors. The main purpose of the Extract-Transform-Load (ETL) processes is to move the data from a source to destinations. In IoT case the source is the sensors, the destination is a storage solution, served by the Cloud layer.

The first part of an ETL job is Extract. The extract is responsible for the connection with the source system, and the data ingestion from there. The input data stream has some important characteristics, which have to be considered: the velocity, the volume, and the variety of the stream. These characteristics have to be counted during the design of the ETL job [34]. The following stage it the Transform: here the data transformed into a suitable format from the destination point of view. The type of transformation depends on the type of the ETL job, and use case. Some usual transformations are the following: data filtering, aggregation, join, concatenation, or String manipulations, or even basic calculations, data cleaning. These transformations can be chained together into a pipeline, for instance: filter out the needed data, and do some calculation on the remaining (filtered) part. Whit the chained transformation really complex job can be created. The last part of an ETL process is the Load to the destination system. In IoT ETL jobs it is critical since meager data arrives in milliseconds, which could be a high load, and burden to the data storage system.

From the QoS aspect, there are some important things to keep in mind in an IoT ETL process. First is the latency: as it was written above, the data arrives in a rapid way, so the process has to handle it without any latency. Another important thing is the scalability: the system has to be scalable in order to expend it easily on demand. The transformation has to be repeatable, observable, in order to keep the data pipeline traceable.

2.1.4 IoT Data Storage

In most of the IoT use-cases, the incoming data has some significant qualities. It has a predefined structure: in a production machine there are a finite number of sensor, and these are changed very rarely during the robot lifetime. However, we have to handle if a sensor failed and it does not send the signal. Another important attribute is, that the data arrives in less than a minute in relatively small chunks: usually around 10-100KB, which is an intense writing load for the data storage. Due to these individual traits, we

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

have to find an applicable storage solution for the IoT use-cases.
Within the Fog architecture, the data is mainly stored in the Cloud layer. As it was described before, the cloud layer is the third layer of the system. The cloud layer is connected to the fog layer and consumes the data from there.
There is two common way to organize storage in the cloud layer. First is the data lake [35], second is the data warehouse [36]. There is some difference between these two approaches: data lake can handle unstructured, raw data, while the data warehouse stores only structured, processed data. For instance, with data lake we have the opportunity, to connect new production robot to our system without knowing its exact data structure. The data warehouse stores structured data, therefore it is ready to use in business intelligence application, or for data visualization. The data quality is also can be better with well-defined data structure. However, a fix structure also can be a drawback, since it can not be easily changed. The other difference between the two technology is the cost. The data lake could be way cheaper than the data warehouse, due to data lake technologies mostly open source, compare to the data warehouse license pricing model. Data lake technologies designed to run on low-cost commodity hardware to keep the operating budget as low as possible. The next difference is about the scaling: data lake use horizontal scaling, while data warehouses use vertical scaling. In horizontal scaling we add one more node in our cluster, in vertical scaling we add more resource - for instance more memory - to our server instance. The horizontal scaling is often easier due to one machine limited capacity. Broaden to the capacity reached often brings downtime or unreliable computational power. The next difference is connected to the architecture design: data lake stores one table separately on different server instances, so the queries distributed between the nodes, while data warehouses store it in one table, which often occurs high concurrency. Behind these two organization aspects, there are different database technologies. One is relational (SQL), the other is the non-relational (NoSQL) database. We have to decide, which one is more suitable for our use-case. In the following, we will compare these two approaches.

Relational databases
The relational database is a digital database based on Relational Data Model (RDM). RDM was firstly introduced C.F. Codd, in 1970 [37]. RDM is based on relation which is a subset of the Cartesian product of sets. Relational databases commonly referred to as SQL databases, which is the used query language. SQL was developed to query and operate on data in databases that follow RDM. In relational databases, the data is organized in tables and rows. The items compliance a predefined schema, with relationships between them. One important difference between the two approaches is the ACID properties. ACID refers to Atomicity, Consistency, Isolation, Durability. It is a set of properties that guarantee transactions reliability. Atomicity is the way we handle the transactions: either all of the transactions are executed, or none of them. A transaction cannot be partially completed. Consistency ensures, that the data in the database is consistent and only the fully successful transaction has an impact. Another important thing about consistency, that none of the defined constraints can be violated. With isolation, we want to ensure, if more than one transaction is executed in parallel, than all of them will be

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

treated as it is the only transaction in the system. For instance, one transaction cannot read data from another, which is not yet completed. Durability ensures that the outcome of a successfully completed transaction will be held in the database.

Non-relational database
Non-relational databases are any database which does not follow the relational data model. They are commonly referred to as NoSQL: not only SQL. This is implied, that the used query language is not the SQL. The biggest difference between these two database systems, that there is no predefined schema in a non-relational database, therefore it is suitable for unstructured data. The database without schema provides high flexibility, since it can store information in a variety of formats. This is one of the key reasons is why non-relational databases are emerging in the past few years. There are three most commonly used data model in non-relational databases: key-value stores such as Redis [38], document stores such as MongoDB [39], extensible record store such as HBase [40]. In every scenario, there is a key, - similarly to the relational case - which allows access to the values. The difference between the data models is how the value is stored. For instance, in the document store, the data is stored as an object such as a binary encoded JSON (BSON). In the extensible record store, the data is stored in rows. In this case, each row have a predefined column groups, which are stored together - in the same node - and as many columns as needed. In a non-relational database, the ACID properties cannot be ensured easily. Other difficulties, that there is no foreign key in NoSQL databases, therefore the analytical queries could be difficult.

As can be seen, different technologies have various abilities and advantages. In the 7.3 Section we tried to find the best storage solution for our industrial use-case.

2.2 Cyber Physical Systems

In the previous section we described the huge potential about the IoT systems, however, in order to make a most of them, we would need appropriate software capabilities. With Cyber-Physical Systems (CPS) we can combine the physical world - the things - with computing capabilities from the cyber world [41]. The cyber-physical systems enable the cooperation between software applications and physical devices in real time, in order to control, monitor, and coordinate the system. To serve these three tasks the software applications are capable to connect to the surrounding physical word, access, collect, and process the available data. We analyse this data and based on the results we can control the system through software services [4]. For a great implementation, it is not enough to understand the physical, and cyber world separately, we have to understand the interaction of both worlds [42]. CPS can be seen as a communicating and computing core in order to create a bridge between the cyber and physical world. The combination of physical and cyber worlds has growth potential in many industries, such as telecommunications, health, and medical equipment, intelligent buildings, etc. Cyber-physical systems have also played a major role in the development of manufacturing processes, which also helps to improve manufacturing competitiveness [43]. This technology may lead the industrial

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

transformation from Industrial 3.0 to Industry 4.0. However, before we start using the cyber-physical system widely, we have to make sure, that the system is reliable, safe and secure, as we described before in the motivation part.

As a recent technology, CPS also suffers from the lack of existing and well-defined structure and methodology for the implementation in the industrial environment. J. Lee et al. [44] proposed a 5 level CPS structure, namely 5C architecture. This architecture can serve as a guideline, during the development of a cyber-physical system for manufacturing application. In general, we can define two main functional elements within the Cyber-physical systems. First is the connectivity to the devices, which enables the real-time data consumption from the physical world and the sending of control instructions from the cyberspace. The second element is the computational capability, that allows us to build powerful data management and analysis for the incoming data. However, these two blocks, are just too abstract to help during the implementation, therefore the 5C infrastructure from J Lee et al. [44] gives us a more clear definition of the architecture.

The 5 components of a CPS system:

- **Connection to the devices:** This component connects to the devices and acquires data. The data has to be accurate and reliable, in order to be usable in the following components. In this component, we have to deal with various types of data sources, and data structures as well. For instance, the data source can be a PLC, what is connected to a sensor, or an industrial PC, which is responsible for multiple PLC's.

- **Data-to-information conversion:** In this component, we have to extract information from the incoming data. This can be done by various algorithms. If we chain multiple algorithms together, we get a data pipeline. The algorithms can be just a simple filtering logic, or more advantage machine learning algorithms too. In the ETL section, these data manipulations will be explained in more detailed.

- **Cyber:** The cyber component is a central information hub of the system. It stores all the information from the connected endpoints, therefore we can get status information about individual machines. With the analysis of this information, we can predict possible future events. For example, we can compare the current state of the machine with the historical data, or compare machines with each other. From the comparison, and analysis we can find similarities and patterns which are the core elements of a predictive algorithm.

- **Cognition:** The user interface of the processed information and machine status. It presents all the available knowledge about the assets to the end-users such as operators, engineers, and managers. Detailed information graphics supports decision tasks, such as maintenance, or process optimization.

- **Configuration:** This layer enables the configuration of the machine from a centralized human interface.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.2.1 Use Cases

Smart Factory
As it was written in the Motivation section 1.1, we can build many applications in a factory with CPS systems. The main advantage of these application is the saving of cost, time and raw material. In a Smart Factory, where we collect data from all the robots, conveyors, and production machine we have the ability to improve the processes. We can also improve product quality, since we continuously monitors all the production parameters, therefore if any conditions reaches a critical level we perform an alert to a corresponding worker. Another important benefit, that we can optimize the supply chain: since we have real-time data about the production, we can predict more accurately how many material we need.

Smart Parking
Smart Parking system can detect whenever a parking slot is taken or not via sensors. They placed one or more sensors in each parking slot. Then the sensor data is collected by a central application.The end-users can track the parking information via a mobile application or in a web page. In this use-case, the fog nodes collect, transform and filter the information from the sensors and send towards the cloud servers. The cloud nodes serve as a data storage, and service provider for the external User Interface. This system helps the better utilization of the empty parking slots since it can be used in private and public parking places and city garages.

2.3 Service Oriented Architecture

IEEE defines software architecture as follows: *"fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution"* [45]. Other, simpler definition: *"Software architecture describes the system's components and the way they interact at a high level"* [5].

Service Oriented Architecture (SOA) enables interoperability among services from different sources. The SOA is a software design principle, with the advantage of flexibility in service implementation. Open Group defines SOA as follows:

"Service-Oriented Architecture (SOA) is an architectural style that supports service-orientation. Service-orientation is a way of thinking in terms of services and service-based development and the outcomes of services" [46]. As can be seen from the definition and from the architecture's name as well, the core components of the architecture are the services. The service is software elements, implements certain functions, with a specified outcome. For instance, a function is to read the value from a temperature sensor. In the SOA system, the service's implementation is separated from the interface. A service is a "black box" for the consumer and this is, what enables the interoperability between services. The consumers and producers do not have to know the exact implementation details in order to communicate with each other, it is enough, to know the other's supported interface or requests type.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

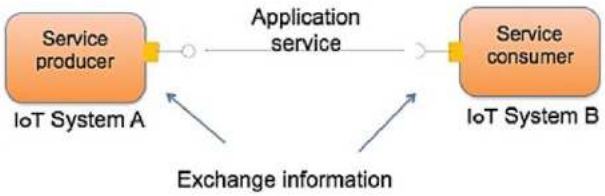


Figure 2.4: Information exchange between Producing and Consuming systems [47]

As it was described in the IEEE definition, a software architecture defines some principles of its design [5]. SOA has some unique characteristic, which is important to understand, in order to implement it in the most effective way:

Discoverable and Dynamically Bound
When a service consumer needs a particular function, it discovers the system to find producers, which can fulfil the consumer's requirements. The search - service discovery - is done via a service registry, which contains all the necessary information about all the available services within the system.

Self-Contained and Modular
In the SOA architecture, the modularity is very important. Modularity means, that a service support set of interfaces, which defines how they will interact with each other. Modularity ensures the interoperability between individual services.

Interoperability
The main purpose, why we are using service-oriented architecture because we want to connect services from different sources. In SOA, each service provides an end-point, an interface for instance. Interoperability means, that this interface is supported by every component, and the services are able to communicate with each other. In order to accomplish the interoperability, we would need techniques for instance mapping between the platform-specific characteristics to a mediating specification.

Loose Coupling
Loose coupling is a way of designing the structure of the system. Loose coupling indicates a few well-known dependencies. This means, that the individual components have no or just a little knowledge about a separated component. This is beneficial when independent teams developing services for the same platform since they do not have to know about others work.

Location Transparency
Location transparency means, that the producer can dynamically change the location, without the client's knowledge. The service consumer knows the location of a provider after a service discovery from the service registry. This could be very beneficial, since, with a load-balancer application we can deploy the same

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

service to multiple locations, in order to have a more scalable, system with greater availability.

Composability
Composability means, that the developer can use existing modules during the development of a new service. This improves the quality of the code since the shared modules are tested.

Reusability
Service reusability means, that the implemented logic in service is independent of technology, and business process. This ensures, that the service can be used in another use-case as well.

Non-functional requirements
Non-functional requirements define how the system should behave in certain situations. These requirements are often called Quality of Service requirements.

2.4 Arrowhead Framework

The Arrowhead Framework [6] is built based on the SOA fundamentals, which were described before. The project started in 2013 with 78 partners, coordinated by Artemis CoIE [48]. The vision of Arrowhead is the enabling of collaborative IoT based on automation. Arrowhead consists of Systems and Services [49]. Service - in Arrowhead Framework context - is used to exchange information between the provider, and consumer systems. A service is produced by a system, in order to implement some functionalities. In a service, multiple capabilities coupled together. In Arrowhead, Systems refers to Service Provider System and Service Consumer System. These systems are not just endpoints, but they can be realized by devices. For instance, a temperature sensor can be a system, which provides the value of temperature as a service. In the framework, there is no pre-defined connection between systems. In Arrowhead there are three types of systems (see Figure 2.5): mandatory core system(I.), support core system(II - III.) and other systems(IV.). These systems will be described later.

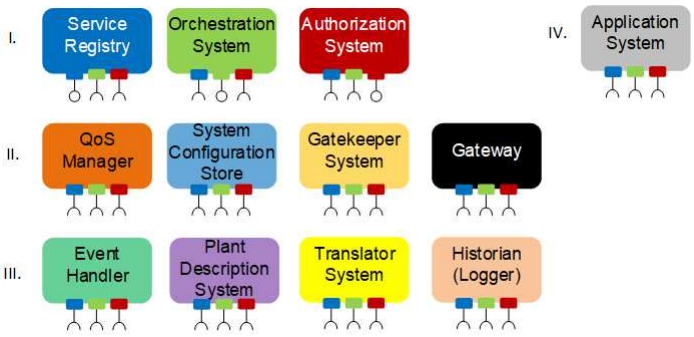


Figure 2.5: The mandatory, optional Core Systems and other Systems [50]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

There was three main focus during the implementation of Arrowhead architecture:

- Provide real-time functionalities
- Strong security for devices, system, and network
- Easy development

In order to implement a minimal working SOA framework based on Arrowhead, we have to implement a minimal set of systems and services. These systems and their services are the Mandatory Core Systems. These systems form a System-of-Systems or Arrowhead local cloud. The local cloud architecture is very important in some special tasks, which has requirements for real-time, safety or maintenance. A local cloud has the advantage of it can be closer to the actual application, therefore it avoids the latency. Another important advantage of the local cloud, that the sensitive data does not have to leave the area, such as a factory, and this could very important.

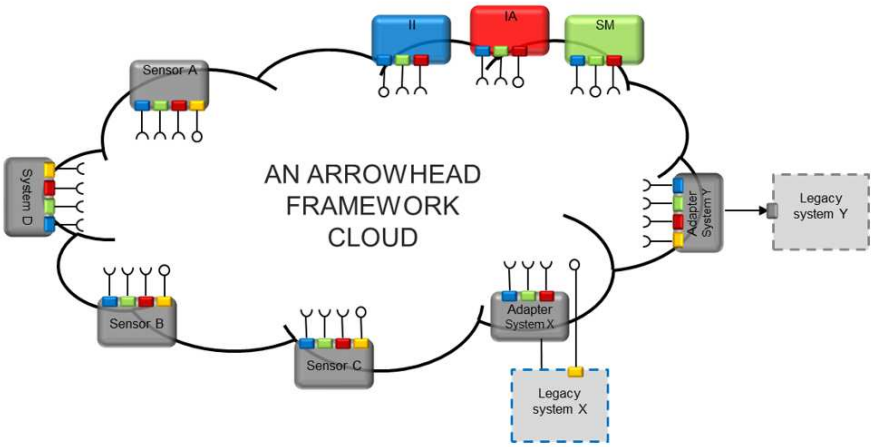


Figure 2.6: Arrowhead Framework System of systems [51]

As it was described before there are mandatory Systems in Arrowhead Framework. The functionalities of the mandatory Systems are:

- Registration and deregistration of services
- Discovery services provided in the cloud
- Authorization, and authentication, and encryption between the communication of two services

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

There are three core modules: ServiceRegistry System, Authorization System, Orchestration System.

Service registry:
Service registry handles the registration and revokes of the different Systems within a local cloud. The systems and their services are stored with their HTTP endpoint and name. A system registration can have metadata, in order to describe the service in more detail. Service registry inserts the data about the systems into a database. All Systems within the framework can discover offered Services via Service Registry's Service Discovery Service. Service registry periodically cleans the database, in order to delete the invalid information.

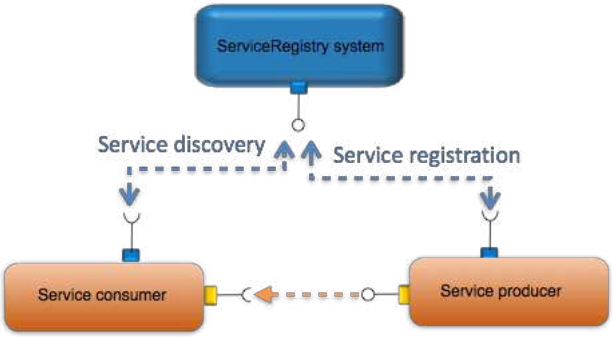


Figure 2.7: Arrowhead Framework Service Registry [47]

Authorization System:
Authorization system stores the inter-cloud, and intracloud access rights. If an Application System needs an authorization token the Authorization System is responsible to issue it. Authorization System stores different pieces of information about the Application Systems, such as the authentication information, access rights (in inter, and intracloud as well). The Authorization System tells if a Consumer System can access a Provider. It uses Transport Layer Security (TLS) [52] with X.509 [53] standard and RSA encryption [54].

Orchestrator:
The Orchestrator System is responsible for the allocation of the Service Providers to the Service Requests sent in by Systems. If a System wishes to consume or produce some kind of service it has first required it at the Orchestrator. This procedure is called the orchestration process in the Arrowhead Framework. First, an Application System sends a Service Requests Form to the Orchestrator, which describes the requested resources, and how the resource will be requested. The end of a successful orchestration process is a response from the Orchestrator system, with the information about the connection, and other necessary parameters.

Custom service:
In order to use Arrowhead framework in different use-cases, we have to implement custom services. In Arrowhead Framework we can implement custom services with the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

modification of client skeleton services [55] [56]. As it was described, the client can be implemented in different programming languages, for instance, Java, C++. There are two types of services: publisher and consumer. The publisher service publishes events to the Arrowhead core system. Each event has a certain type. Customer service can subscribe to one, or more types of message, and consume it from the Arrowhead system.

The main advantage of the SOA framework, that we have the ability to integrate various services. All the different service serves a function in the system. With SOA we can implement our custom function, or either we can use an existing service. However, the deployment of the heterogeneous services is not a straight forward task. As it was mentioned in the Motivation section 1.1, we have to deal with independent teams, and various programming languages.

2.5 Containerization

The main purpose of the containerization is to help the deployment and continuous delivery of SOA software framework into the CPS/IoT systems. The biggest advantage of the containerization is the ability of the encapsulation the application with all the necessarily components into a a self-contained software unit, called container file. A container file includes all the source code, configuration files, dependencies, and runtime environment. The container provides a portable, lightweight, scalable solution for the deployment, and management of the heterogeneous services, which is one of the problems of the CPS/IoT systems [57].
Containerization is a virtualization technique with the aim of providing an interoperable platform for the applications. The foundation of the containerization is the Linux operating system level virtualization mechanism, called Linux Container (LXC) [58]. LXC allows the separation of processes, with isolated namespaces, and control groups for the control and management of resource limits.
Containerization is similar to the Virtual Machines (VM) [59], where we emulate the full computer system. However, there are some key differences between these two technologies. The main difference is the abstraction layer. Container-based systems uses operating system level virtualization. This means containers are an abstraction of the application layer, which enables the deployment of multiple isolated containers within the same OS kernel. In the other hand, virtual machines are the abstraction of physical hardware, using full virtualization, which transforms one machine into several machines via the hypervisor. For this purpose, each VM have a full copy of an operating system, applications, and all of the necessary files. This makes VMs use more disk space and memory. The VM utilizes the standard boot process, which also makes it slower, than containers. In general containers more portable, and demands significantly less computational resource.

2.5.1 Docker

Docker [60] is one of the most common application containerization technology, based on LXC mechanism. It is a container based platform for application deployment and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

runtime environment. A container is the smallest unit in the Docker environment [61]. Docker supports all the major Linux distributions and Windows operating system as well. Docker also supports different hardware architectures, such as x86_64 [62] CPU architecture, which is commonly used in the PCs, and ARM architecture [63], which for instance, can be found in Raspberry Pi. Docker also provides isolation from the other applications, and host machine's software infrastructure, which makes it more secure. The isolation also supports the deployment of multiple version of an application without conflicts. Besides that docker being secure, it is lightweight, due docker uses OS system kernel.

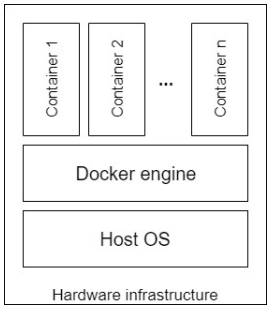


Figure 2.8: Docker architecture

2.5.2 Docker Swarm

Docker has a cluster management and orchestration feature, called swarm [64]. Docker in swarm mode consists of a group of machines what are running the Docker engine and joined into a cluster [65].

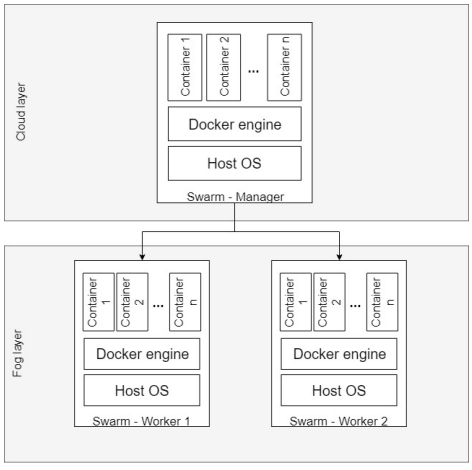


Figure 2.9: Swarm architecture

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The container in the swarm is called service. Docker swarm has a declarative configuration model, where we can define the desired state for the services. In this model, we are able to define among others, the number of replicas from the service, or the minimum CPU, and memory requirement. The swarm maintains this optimal state for the service with Orchestration function. These facilities make the swarm cluster scalable, reliable, secure and fault-tolerant, which supports the QoS requirements. For instance, if we want to scale our application, we can deploy multiple replicas from it. Docker swarm uses Round Robin algorithm to arrange external load balance between deployed replicas by default, but we have the ability to employ an internal load-balancer as well. The internal load balancing in swarm done by the ingress routing mesh, which allows that if a service is deployed to swarm, and published a port, then any node in the cluster will be able to accept the incoming request, even if there is no service instance on that node. For instance, we deployed an Arrowhead service on our architecture, which consists of a master node, and two worker nodes, with `node.role==worker` placement constraint. In this case, we do not have to know, where will be our service deployed, we can send a request to every node in the cluster. This process can be seen in the Figure 2.10.

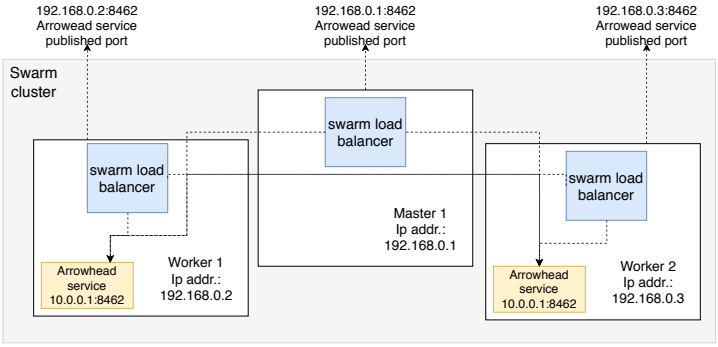


Figure 2.10: Load balance in Docker swarm

With the swarm, we can update the service during the runtime, for instance, we scale it up or down on demand. The system is fault tolerant, which means if one of the services is failed, or the manager detects a node outage, then the swarm manager updates the service, or place it into another suitable node. However, if the manager node fails, then we would need to create a new cluster to continue our work, therefore Docker suggests to deploy more than one manager servers in production. With multiple manager instances, swarm uses the raft consensus algorithm, and a distributed state store, where Docker stores the information about the cluster.
Docker also grants control about the service's resource usage. We have two types of limit: first defines a constraint for the maximum usage, second specify a minimum available resource in the node. For instance, if the service has a maximum limit as 50% CPU, and reservation with 10% of CPU, than Docker, will not let the container consumes more than 50% of CPU, but the orchestrator only schedules it into a node with more than

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

10% available CPU. The communication within the swarm cluster is secure: each node uses TLS authentication and encryption for the secure transmission.

2.6 Continuous integration and deployment

As it was described before, in a CPS/IoT system we have to integrate various services from different sources, implemented with diverse programming languages. During the integration, we must test these services, if they work as expected, and they do not interfere with each other. After the test phase, we have to be able to deploy these services into the heterogeneous nodes in the system. In order to do these actions, we implement a Continuous Integration and Deployment (CI/CD) solution. A CI/CD system takes three components:

Continuous Integration (CI)
Since the development is done by independent developers, we have to have a strategy to merge their changes together into a master version. CI provides a practice, how these changes can be integrated. During the integration process, the changes are validated by automated tests, in order to avoid errors.

Continuous Delivery (CD)
CD is an extension of the CI process, in order to make the new version of the service available as quickly as possible in a sustainable way. Besides the automated tests, in Continuous Delivery, we employ an automated deployment process. This process ensures the deployment of the application at any time by a trigger event.

Continuous Deployment (CD)
The difference in Continuous Deployment, compared to Continuous Delivery, that there is no human interaction during the workflow. This means the developers publish their changes, and the Continuous Deployment pipeline automatically runs all the pre-defined tests, and build an artifact from the source code. In the case of successful tests, the process automatically deploys the service into the production environment.

Continuous Integration and Delivery is a complex process, with a few necessary components which can be seen in Figure 2.11
The first step of the process is the implementation of the application, and upload the source code into a version control system. We use Version Control System (VCS) to keep track of the software changes. With VCS, we can enable the collaboration of teams or team members. After the shipment of the source code into a VCS, we apply several types of test. The first type is the unit test. During the unit test, we want to ensure, if the individual units of service work as expected, and the application can perform its functions. After a successful unit test, we perform an integration test. With this type of test, we want to ensure, if the application can work together with its runtime environment and the other applications without interfering them. The main purpose of these tests is to detect software failures before the deployment process. After the integration test, we containerize the services and apply the acceptance test. During the acceptance test,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

we test if all the specified requirements for the application are met, and the software is acceptable for delivery. The last step of the process is the deployment. As can be seen in the Figure 2.11, in Continuous Delivery the deployment process is manual, while in Continuous Deployment it is fully automatic.

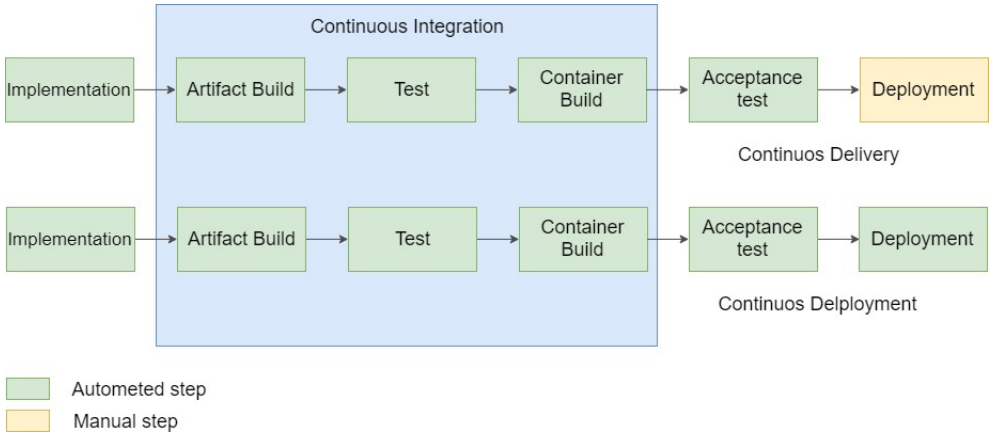


Figure 2.11: CI/CD Process

The adaptation and appropriate implementation of the CI/CD process is very important due to some aspects. One of the most important aspects of the CI/CD tools is to avoid the deployment of the services with errors and failures. As it was written before, we use automated tests, in order to achieve less error in services. With a continuous integration job, we can define different types of tests, and assign priority to them. In case of a test with high priority is failed, then it has to be corrected immediately. As it was written before, in a Pipeline we can define stages, therefore if an automated test failed, then we are not going to deploy the service to the cluster. Additionally, from the test data, we can observe patterns about failure frequency or length of the process.
The second aspect is the metrics collected during the workflow, such as test coverage, or code quality. There are many tools to measure the code quality based on predefined criteria, for instance: software size, maintainability, security, code duplications. One interesting metric is the cognitive complexity of the source code: this measure, how hard to understand control flow of the code [66]. Most of the software quality measurement tool supports to add domain specific rules, which means we can define additional, more suitable conditions. The integration of measuring services means, that if the software cannot reach a certain limit of quality or test coverage then it failed to build. This also ensures us, that the service with an error or inadequate quality cannot reach the production environment.
Another important aspect is the faster and easier deployment of the services into the cluster. As it was written in the Problem statement section, we have to deal with various programming language versions and dependencies. In order to overcome this problem, we employ a container based deployment approach. With the container based approach

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

during the building phase, we are not just building an executable artifact from the source code, but we create a container image as well. Since we encompass the source codes of the services with their dependencies, and runtime environment, we do not have to install them manually into the nodes within the cluster, which often occurs many problems. For instance, we installed a wrong version from the programming language, or we have not installed a necessary dependency into a node. With container based deployment, we have more insight and control over the versions of the software dependencies. Also difficulty during the deployment process, that a high dependable CPS/IoT system, cannot be stopped if a build failed. In such cases, we can use the latest successful version of our service. However, if a test failed, the developers have to correct the code. The developers are responsible and accountable for their work, due to the user management technologies used in CI/CD tools.

2.6.1 Jenkins

In this work, we use Jenkins [67] for the implementation of the CI/CD workflow. Jenkins is an open source automation server written in Java. With Jenkins, we can cover all CI/CD activities, including project building, testing, code analysis, and deployment of the services.
In order to define a CI/CD workflow in Jenkins, we specify a Pipeline. A Jenkins Pipeline is an automated expression and description of the CI/CD process operation. In a Pipeline we chained actions together, for instance getting source code from version control, or testing the service. The Pipeline is written in a text format with domain-specific language syntax. The specification file called Jenkinsfile. The main advantage of the file-based specification is that we can add this file to the project source code, and it can be versioned, and reviewed such as any other file in the project. In a Pipeline, we can define stages, which are the subsets of the entire CI/CD task, such as building the application. If one stage is failed - for instance, due to a failed test- then the whole workflow will stop, which prevents the deployment of services with errors. The detailed workflow will be described in the section 5.3.

2.7 Runtime Monitoring

A CPS/IoT system is a complex structure, with continuous interactions between components and services. During these interactions and operations, the system generates diverse behavior. Runtime monitoring is a technique for the observation, verification, and evaluation of the system behaviors, whether they are consistent with a given correctness specification, or not.

As we explained in the description of the CI/CD pipeline, every service is tested before the deployment into the production environment, however, testing is not feasible. According to the study from Buttler and Finelly [68], we would need to perform too many tests (e.g 10^8 hours of test) to achieve a reliable, fully fault-tolerant system. Therefore we use a runtime monitoring solution, since, it can observe a system behavior at runtime, detects

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

and reacts on the inconsistent behavior and it can help us to achieve a more reliable system.

There are two types of runtime monitoring systems: offline and online monitoring. The main difference between these two approaches is the execution time. In the offline monitoring, we evaluate and analyze the data of a recorded execution trace after the system terminates. Offline monitoring tools mainly used for software debugging and analysis purposes. From the global analysis of the execution history, we can identify patterns about software or hardware failures, which helps us to improve the software quality. In the other hand, online monitoring dynamically monitors the system during the runtime. With online monitoring tools, we can verify the system correctness dynamically in real-time. For the online monitoring system, we can distinguish various strategies. These strategies depend on placing or the employed protocol.

The runtime monitoring system, in general, contains two mandatory components. The first component is a system behavior collector, which is responsible for the collection and measurement of all the relevant information and variables of the system. The second component is correctness analyzer, which validates the collected information based on the system specification. The monitored system commonly referred to as "system under observation". If any requirement violation occurred in the SUO, the monitoring system performs an alert, or reconfigure the system.

Besides the mandatory components, we can distinguish between various implementation strategies for monitoring system components, depending on the placing, and employed protocol. One placing method is the non-intrusive runtime monitoring, where the monitor is separated from the application. In non-intrusive case, the monitoring system treats the service as a black box, therefore we do not have any internal information about the service inner status. The other placing method is the intrusive, where we include the monitoring code into the source code of the application. With the intrusive technique, we have more insight about the internal performance of the system. With intrusive monitoring, we can measure for instance, how many messages are sent by the service. We can categorize the used protocols based on the connection between the monitor and the system. In one side we have the loosely-coupled approach with completely asynchronous monitoring, while on the other side, we have the tightly coupled approach with completely synchronous monitoring. In the completely synchronous case, if any violating event occurred in one component of the system then the monitor interrupts the complete system. Since synchronous monitoring provides real-time detection, it suitable for a highly dependable system. In a completely asynchronous case, the system receives the system behavior in the background, and decide the possible actions without system interruption. However, asynchronous monitoring takes less computation resource, they cannot be used for real-time decisions, due to the late detection.

This section described the architecture of a typical monitoring system. However, we also need a system specification, which describes the optimal behavior for the system. The monitoring system then interprets this specification, and classify the observed system behavior. One of the system specification tools is the Temporal Logic language. In the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

following section we are going to demonstrate the Temporal Logic language, and give some example of the usage.

2.7.1 Temporal Logic

For the system requirements specification, formalism, such as temporal logic is commonly used. It can specify the expected behavior in a declarative manner, which can be easily translated into monitoring rules. With temporal logic, we can specify properties over time. Linear-time temporal logic (LTL) [69] is a language to specify logic (sequences), and their relations over time. The monitoring system than observe those sequences, and verifies if any violation occurs. While digital systems generate discrete sequences of logical states, the physical system's behaviour can be seen as a continuous signal. In order to use LTL in context of cyber-physical systems, it should be extended, to be able to check continuous signal. Signal Temporal Logic(STL) [70] is an extension of LTL with moving from discrete to dense time (such as in the MTL/MITL [71]), and the usage of predicates on numerical values.

We adopt the following conventions from [72]. Signal: a signal is a function $T \to E$, where T is an interval of $\mathbb{R}^+$, and $E \in \overline{\mathbb{R}} := \mathbb{R} \cup \mathbb{B}$. If $E = \mathbb{B}$, than the signal is a Boolean signal, else it is a real-valued signal. An execution trace w is a set of real-valued signals $\{x_1^w, \ldots, x_n^w\}$ with time domain T of $\mathbb{R}^+$. We denote by X and P finite sets of *real* and *propositional* variables. For a given variable $v \in X \cup P$ and a signal w, $\pi_v(w)$ stands for the projection of w on its component v. We consider the STL logic with both future and past operators. The syntax of STL formula φ is defined by the grammar:

$$\varphi := p \mid x \sim c \mid \neg p \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathcal{U} \varphi_2 \mid \varphi_1 \mathcal{S}_{\mathcal{I}} \varphi_2$$

where $p \in P\{p_1, \ldots, p_n\}$ *propositions*, $x \in X$ *real variables*, $\sim \in \{<, \leqq\}$, $c \in \mathbb{Q}$, *and* $I \subseteq \mathbb{R}^+$. The STL semantics of satisfiability relation of signal w at time point t defined as: $(w,t) \models \varphi$. The interpretation of logic only over finite traces $t \in T$

$$\begin{aligned}
(w,t) &\models p && \leftrightarrow \pi_p(w)[t] = \text{true} \\
(w,t) &\models x \sim c && \leftrightarrow \pi_x(w)[t] \sim c \\
(w,t) &\models \neg\varphi && \leftrightarrow (w,t) \not\models \varphi \\
(w,t) &\models \varphi_1 \vee \varphi_2 && \leftrightarrow (w,t) \models \varphi_1 \text{ or } (w,t) \models \varphi_2 \\
(w,t) &\models \varphi_1 \, \mathcal{U}_{\mathcal{I}} \, \varphi_2 && \leftrightarrow \exists t' \in (t+I) : (s,t') \models \varphi_2 \text{ and } \forall i < k < j, (s,k) \models \varphi_1 \\
(w,t) &\models \varphi_1 \, \mathcal{S}_{\mathcal{I}} \, \varphi_2 && \leftrightarrow \exists t' \in (t-I) : (s,t') \models \varphi_2 \text{ and } \forall i < k < j, (s,k) \models \varphi_1
\end{aligned}$$

To build complex formulas, we can use temporal and Boolean operators. The basic temporal operators are:
$\circ p$: p is true in the next moment of time $(t+1)$
$\Box p$: p is true for all future moments
$\diamond p$: p is true in some future moments
$\varphi \mathcal{U} \Psi$: φ is true until Ψ is true

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

We can use Boolean operators ($Not(\neg)$, $And(\wedge)$, $Or(\vee)$, $Imply(\rightarrow)$)
Example:
We have two propositions in our system:
- The highest CPU usage can not be higher than 70%
- The available CPU can not be lower than 5%

$$(p < 70\%) \wedge (p > 5\%)$$

In a heterogeneous CPS/IoT system, where discrete, and continuous signals also involves, the qualitative binary *yes* or *no* may not enough in every application. In such systems we would want to specify some degree of tolerance for instance to initial conditions, or system parameters. Fainekos and Pappas [73] proposed an extension for STL, which deals with quantitative semantics. In their work, they replaced the binary decision with quantitative robustness degree, which returns a real value, which indicates how far is a signal from the specification.

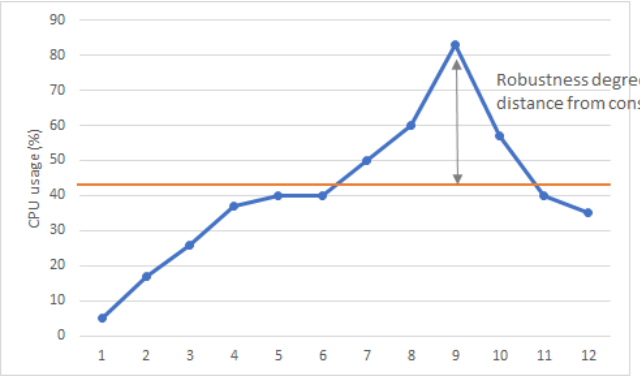
We can define the robustness degree function $\rho(\varphi, w, t)$, where φ is an STL formula, w is a signal, $t \in T$ is a time instant.

$$\rho(\varphi, w, t) = \begin{cases} \infty & \text{if } \pi_p(w)[t] = \text{true} \\ -\infty & \text{otherwise} \end{cases}$$

$$\rho(x \sim c, w, t) = c - \pi_x(w)[t]$$
$$\rho(\neg\varphi, w, t) = -\rho(\varphi, w, t)$$
$$\rho(\varphi_1 \vee \varphi_2, w, t) = max\{\rho(\varphi_1, w, t), \rho(\varphi_2, w, t)\}$$
$$\rho(\varphi_1 \mathcal{U}_{\mathcal{I}} \varphi_2, w, t) = sup_{t' \in (t+I) \cap \mathcal{T}}\ min\{\rho(\varphi_2, w, t'), inf_{t" \in (t', t)}\ \rho(\varphi_1, w, t")\}$$

Where ρ is a quantitative satisfaction function of time, which returns a real number. If we consider $x \in X$ real variable and $c \in \mathbb{Q}$, $x < c$, the quantitative satisfaction function returns the relative position of x to c. If we negate the signal, the quantitative satisfaction formula returns the negate of the relative position. If we have two signals, the function returns the maximum of the distances between the signals and criteria.

In the Figure 2.12 can be seen the robustness degree:



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Figure 2.12: Robustness degree

Monitoring algorithms with STL

There are many different implementations of the monitoring exist for the different semantics, such as the qualitative monitoring algorithm [74], or the quantitative monitoring algorithm [75]. We adopt the algorithm from [72], which is a simpler form as the original quantitative algorithm.

Algorithm 2.1: Monitor [72]

Input: STL formula φ, input signal w
Output: output signal y

```
1  begin
2    switch φ do
3      case p do
4        CheckSatisfaction(p, w);
5      case x ∼ c, where ∼∈ {<, ≦} do
6        CheckSatisfaction(x ∼ c, w)
7      case ∗φ, where ∗ ∈ {¬, ⋄_I, □_I} do
8        w′ := Monitor(φ, w)
9        CheckSatisfaction(∗, w′)
10     case φ1 ∗ φ2, where ∗ ∈ {∨, U_I} do
11       w′ := Monitor(φ1, w)
12       w" := Monitor(φ2, w)
13       CheckSatisfaction(∗, w′, w")
14   end
15 end
```

In the following, we will define CheckSatisfaction for each case. Since we are working with a CPS system, where the quantitative approach would be beneficial - as it was described before - we will provide a definition for both qualitative, and quantitative approaches as well.

In the following, $p \in P\{p_1, \ldots, p_n\}$ *propositions*, $x \in X$ real variables, $\sim \in \{<, \leqq\}, c \in \mathbb{Q}$, *and* $I \subseteq \mathbb{R}^+$.

case p:
Input:(p, w)
Output:
Qualitative: $y[t] = p$
Quantitative: if $p ==$ true, $y[t] == +\infty$, *else* $y[t] = -\infty$

case $x \sim c$, where $\sim \in \{<, \leqq\}$:
Input:$(x \sim c)$
Output:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Qualitative: $y[t] = (\pi_x(w)[t] \sim c)$
Quantitative: if $y[t] = \pi_x(w)[t] - c$

case $*\varphi$, where $* \in \{\neg, \diamond_I, \Box_I\}$:
Input:$(\neg, w')$
Output:
Qualitative: $y[t] = \neg w[t]$
Quantitative: if $y[t] = -w[t]$
Input:$(\diamond_I, w')$
Output:
Unbounded Eventually $\diamond_{[0,\infty)}$
In this case we have a time interval $I \in [0, \infty]$ than the output signal is:
$y[t] = sup\{w[t]\} = max\ w[t_i]$ where $t \leqq t'$ and $t \leqq t_i$

Bounded Eventually $\diamond_{[a,b)}$
In this case we have a time interval $I \in [a, b]$ than the output signal is:
$y[t] = max\{w[t_i]\}$ where $t \in [t + a, t + b)$
In this case, we have a sliding window for the sequence of signal w, with the size of $b - a$
Input:$(\Box_I, w')$
Output:
The form for globally operator can be deducted as: $\Box_I \varphi \Leftrightarrow \neg \diamond_I \neg \varphi$

case $\varphi_1 * \varphi_2$, where $* \in \{\vee, \mathcal{U}_\mathcal{I}\}$:
Input:$(\vee, w', w")$
Output:
Qualitative: $y[t] = w'[t] \vee w"[t]$
Quantitative: if $y[t] = max(w'[t], w"[t])$ Input:$(\mathcal{U}_\mathcal{I}, w', w")$
Output:
The qualitative algorithm can be found in [76]
Quantitative:
Unbounded Until $\mathcal{U}_{[0,\infty)}$

In this case we have two finite length N signals: $w', w"$, and we have z and z' the robustness signals of φ_1 and φ_2, with $(t_i)_{i \leqq n_z}$ and $(t'_i)_{i \leqq n'_z}$. The output is the robustness signal of $\varphi_1 \mathcal{U} \varphi_2$. The computation of the output signal y can be done by backward induction, and we obtain the following formula:

$y[t] = max(min(w'_k, w"_k), min(w'_k, y_{k+1}), k \in \{0, \ldots, N-2\}$

Bounded Until $(\mathcal{U}_{[a,b)})$
To compute this, we can use the following lemma:

Lemma 1

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

$$\varphi_1 \mathcal{U}_{[a,b)} \varphi_2 \Leftrightarrow \diamond_{[a,b)} \varphi_2 \wedge \varphi_1 \mathcal{U}_{[a,+\infty)} \varphi_2$$
$$\varphi_1 \mathcal{U}_{[a,+\infty)} \varphi_2 \Leftrightarrow \square_{[0,a)} (\varphi_1 \mathcal{U} \varphi_2)$$

Than we can compute the Bounded Until ($\mathcal{U}_{[a,b)}$, $a < b < +\infty$) in three steps:
w_1 :=CheckSatisfaction($\mathcal{U}_{[0,\infty)}$, w', w")
w_2 :=CheckSatisfaction($\square_{[0,a)}$, w_1)
w_3 :=CheckSatisfaction($\diamond_{[a,b)}$, w")
return CheckSatisfaction($\wedge$, w_2 w_3)

We provide an illustrative example in the Figure 2.13 below, for the **case** $x \sim c$, where $\sim \in \{<, \leqq\}$. For instance in $t = 1$ time, the output signal in quantitative case is 0, in qualitative 10. In $t = 4$ time the output values are: 1, -20 respectively.

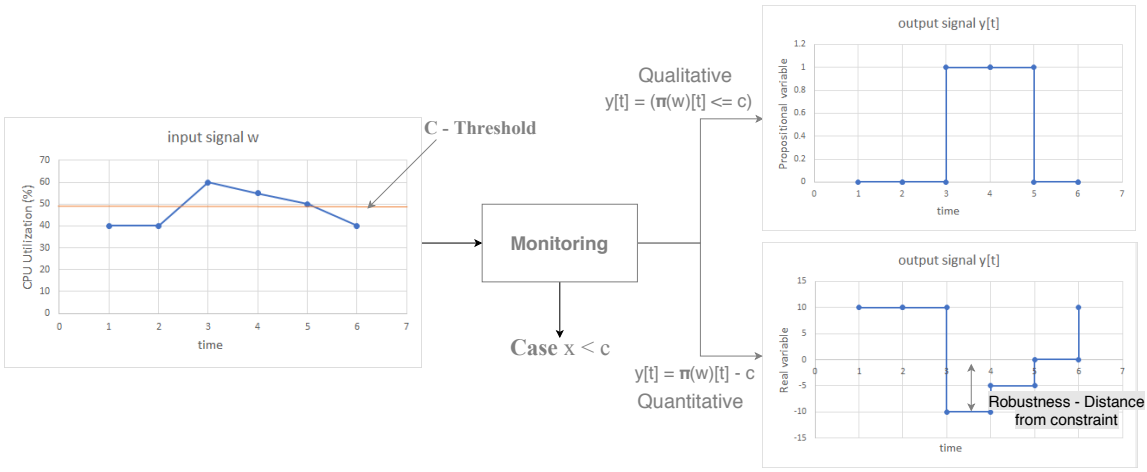


Figure 2.13: STL monitoring

The algorithm beyond is an offline monitoring algorithm. During the offline monitoring, we assume, that we have the entire trace available for our monitoring solution. In a working CPS/IoT system, it is not the case, therefore we have to extend the offline monitoring algorithm, and we will get online monitoring. As in the offline monitoring, we can have Qualitative, and Quantitative types.

One type of Qualitative online monitoring extension is proposed in work [77], called incremental marking. The essential of this idea is that the signal is available in chunks for the observation, and we can only apply the offline algorithm for these chunks.

There are many online monitoring implementation for the Quantitative approach. First is the bounded future, with unbounded past formulas, introduced in this work [78]. This algorithm updates a table with the past and future formulas, which are required to evaluate the formula φ. For the unbounded past formula, the algorithm stores the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

computation of past formulas as a summary and updates this each time, when a new chunk arrives. Second is robust online monitoring of partial traces [79].
Another work deals with unbounded future formulas [80]. Since in such formulas the robust satisfaction intervals are meaningless, they computed the input formula in the restricted available signal. In order to avoid to store the whole signal, or do the same computation again, they store the conducted result as a summary.
In the work of Stefan Jakšić et al [81], they equipped the STL quantitative semantics with weighted edit distance metrics, which quantifies both space and time mismatches. WED provides information about the frequency of the property violation.

2.7.2 Prometheus

The previously described STL language can help us to define constraints and rules in our system. However, the description language is not enough, we need a monitoring application, which is able to validate these rules on the incoming signals, as it can be seen in the Figure 2.13. Although there is a tool for the runtime verification of the STL formulas on the system's signals [82] [83], but it is not suitable for us, due to the lack of connectivity with our architecture. We have chosen the Prometheus monitoring system [84] since, we can integrate it with our system, due to the fact, that Docker introduced an experimental feature from version 1.13, which supports the Docker Engine to export the metrics in Prometheus syntax [85]. Although there are other monitoring systems out there [86].
Prometheus [84] is an open-source monitoring system, and time series database, supplemented by a modern alerting approach. Prometheus is originally developed at SoundCloud, in 2016 it joined to Cloud Native Computing Foundation [87], whose projects also include Kubernetes as an example.
The architecture of the Prometheus system can be seen in the Figure 2.14 below:

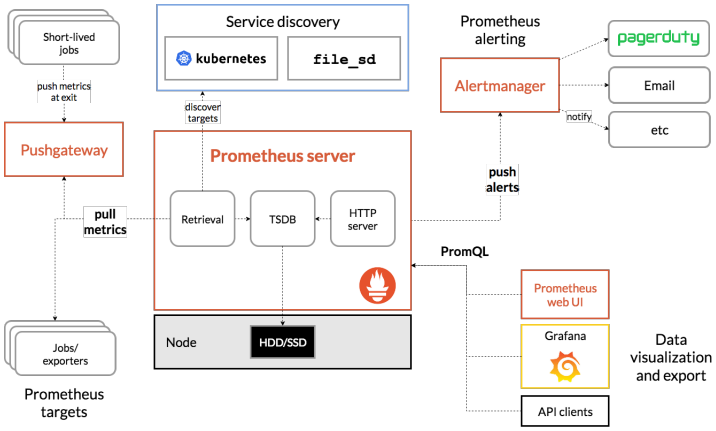


Figure 2.14: Prometheus Architecture. Source: `https://prometheus.io/docs/introduction/overview/`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Prometheus server

The core component of the monitoring system: it pulls the metrics from the job exporters, and store it in a time series database (TSDB). It also provides a user interface (HTTP server), where you can check the status of the job exporters, and the defined alerts in the system.

Prometheus stores all data as a time series, where timestamps increase monolithically ($t_n < t_{tn+1} \forall n \in \mathcal{N}$). Metric values are stored over time (timestamp t, value v)

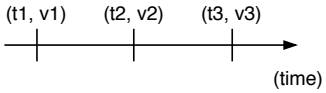


Figure 2.15: Prometheus data model

A time series uniquely identified by its metrics name, and a set of key-value pairs also known as labels. For instance:
Metric name is: `container_memory_usage_in_byte`
Labels: `container_name, instance`
We can see the elements, of the data model in the Figure 2.16 below:

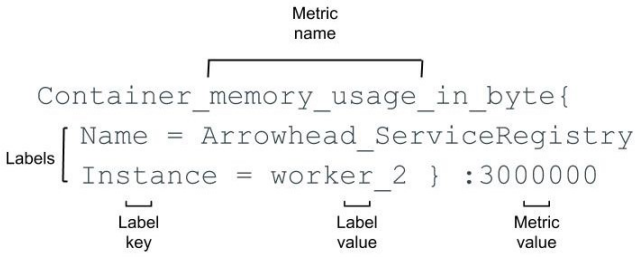


Figure 2.16: Prometheus data model elements

Alert Rules

Prometheus server is responsible for alerting. Prometheus perform alerts based on alerting rules, which are evaluated at regular intervals. The recommended way to define alerting is based on this paper [88], Rob Ewaschuk's observations at Google. We can define a wide variety of rules regarding the system status, and resource limitations. For instance, we can define an alert, if the docker container consumes too much resource, or it gets in a failed status. Unfortunately, we can not use directly the STL language for the specification of the alerting rules, Prometheus use its own expression language for this purpose [89]. Although we can not use STL, Prometheus expression language (PromQL) has similar capabilities with straight-forward syntax. In the following, we provide some example of the usage of PromQL, and the conversion between STL and PromgQL:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Constraint for the container's CPU usage (in percent):
STL:
$\varphi \models G(x[t] < 80)$
Signal Always (G) has to be under 80% of CPU usage.

PromgQL:
```
sum(rate(container_cpu_usage_seconds_totalname="jenkins"[1m])) /
count(node_cpu_seconds_totalmode="system") * 100 > 80
```
The CPU usage of the service "jenkins" divided by the total CPU usage of the node can not higher than 80.

The same constraint for too much memory usage (in bytes):
STL:
$\varphi \models G(x[t] < 120000000)$
Signal Always (G) has to be under 120000000.

PromgQL:
```
expr:  container_memory_usage_bytes{name= "Arrowhead_ServiceRegistry.*"}
> 120000000
```
The memory usage of the container with name of "Arrowhead_ServiceRegistry cannot be higher than 120000000.

If we do not want to send an alert immanently (because there is a possibility for a higher consumption for some time):
STL:
$\varphi \models F(G_{[0,5]}(x[t] > 12000000))$
F means, Eventually (F) at some time t, from that time t the signal value Always (G) -between time 0,5- will be higher than 12000000.

PromgQL:
```
expr:  container_memory_usage_bytes{name= "Arrowhead_ServiceRegistry.*"}
> 120000000
for:  5s
```
This behaviour can be achieved in Prometheus with FOR clause, which causes Prometheus to wait for a certain duration. In this example the duration is 5 seconds.

Alerting rules, if service is down:
STL:
$\varphi \models G(x[t] > 5 \wedge x[t] < 120000000)$
This means signal Always (G) has to be under 120000000 and over 5 bytes.

PromgQL:
```
absent(container_memory_usage_bytes{name= "Arrowhead_ServiceRegistry.*"})
```
Absent function indicates, that there is no time series exist for the metrics. In this case, this means there is no time series for the memory usage, which means the container is down. This Prometheus rule is not identically same as the STL rule, since it lacks the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

$x[t] < 120000000$, but this is because the alert for the container is down, and container consumes too much memory is logically separate. We can combine Prometheus alerting rules with same Boolean operators (and, or).

We can define the alerts in alert.rules file, an example rule would be for a container high memory consumption:

```
– alert: resource_tester_high_memory
expr: container_memory_usage_bytes{name=~"resource_tester.*"} > 99999500
for: 5s
labels:
severity: warning
annotations:
summary: "Test service high memory usage"
description: "Test service memory consumption is at {{ humanize $value}}."
```

As can be seen, besides the alerting expression there are more elements in a rule. We can use `for` clause when we want to wait for a certain duration after the first occurrence of the alert. For example in the case beyond we let the service for 5 seconds to consume more memory than the threshold, which is beneficial during the application startup. Another element is the `annotations` part, which contains additional information about the alert.

Job, exporters

Prometheus has an Agent-based monitoring structure, which means there is an agent in all nodes within our cluster. It results in a more secure system, with less network bandwidth usage. The system use pull mechanism for the metrics collection, which immediately let us know if a service is down. Prometheus is a WhiteBox Monitoring tool, which means it not just provides information about the system, but also sends detailed metrics about the service internal state, for instance, the number of requests for a web server. In order to monitor a docker swarm environment (swarm nodes, and Docker containers), we need two kinds of exporters.
First is the Node-exporter: this is responsible to collect the metrics from the nodes in the swarm cluster, such as resource utilization (CPU, memory, storage, network). For this purpose, we used the official Prometheus Node exporter [90].
Second is the Container-exporter: this is responsible to collect metrics about the individual docker containers. However, as it was mentioned, Docker engine has started to support the metric collection by default, it is not as detailed, as cAdvisor [91]. Container Advisor is developed by Google, with the aim of providing the users more accurate understanding of the resource usage and performance characteristic of a container.

Alert manager

This component is responsible to handle the alerts, and route them, according to the predefined routes. We can define route endpoints for the alerts, for instance, Slack, or

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

HTTP web server.

Grafana

Grafana [92] is a real-time visualization component. It queries the Prometheus time series database and generates graphs from the result. It can automatically refresh the graphs periodically. We can build an informative dashboards with various graphs.

Service Discovery

It discovers the job exporters dynamically. In docker swarm, it is very important since if we add a plus node, Prometheus will be able to monitor it automatically.

2.8 Quality of Service analysis of a CPS/IoT system

2.8.1 Introduction

In the CPS/IoT system we combine many different technologies, devices, protocols from a wide variety of sources. We have a continuously growing number of sensors on the one side of our system and servers, cloud components on the other side. CPS/IoT is an ever-growing field with different use cases from different areas and all these areas require different QOS aspects.
As it was written in the introduction part, ITU-T Rec. E. 800 defines Quality of Service (QoS) as "collective effect of service performance which determines the degree of satisfaction of a user of the service" [16].

Salman et al. [93] focused on QoS reliability in Cyber Physical Sensor Networ (CPSN). In their work, they introduced some QoS design requirements during the implementation of a CPS system:

- Service Oriented Architecture (SOA) for reducing the complexity of the infrastructure, and enable scalable development
- QoS aware communication and networking
- Intelligent resource management for resource allocation such as CPU time, bandwidth and memory
- QoS aware energy consumption

According to these requirements, there are more studies, focused on just a specific topic, such as resource management. In the study from Balasubramanian et al. [94], they implemented a domain-specific modelling language (DSML), which support the specification of per-application CPU and per-flow network QoS requirements. In another study from Dewei Peng and Youlin Ruan [95], a new QoS model was introduced, which

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

is according to the improved Analytical Hierarchy Process (AHP). In this model, they considered the resource state, the user preferences and device performance.

Another aspect is the integration of the IoT with cloud computing [96]. In this topic, there are huge difficulties, for instance, the difference between computational power in IoT and cloud nodes. Another challenge in seamless integration is the network architecture, which needs to support QoS needs such as latency, data security, privacy, reliability, as it was described before.

2.8.2 QoS Quality Models

As it was described, CPS/IoT applications operate in a wide variety of areas with different system requirements [97]. For instance, industrial IoT systems have specific security preconditions, while autonomous driving systems are heavily delay-sensitive. All these applications have different QoS requirements, which depend on various factors. As an example, a water sensor in a smart agriculture application can have a minute-long delay, but a sensor in a car which feeds the self-driving software cannot have such a long delay. IoT applications QoS factors mostly depends on the use-case, but these factors are not independent. By way of example, we have to deal with the trade-off between sensor reliability and service availability. To have an overall picture of the considerable QoS factors for CPS/IoT system we have to define a quality model. Quality models support the comparability between different QoS approaches.

According to the quality model, specified for Internet QoS, we can categorize our system in three distinct categories:

- **Best Effort:** The Best Effort model is also known as no QoS, because it does not implement any QoS at all, therefore this is the simplest model of all three. Best effort does not allow any resource reservation or any kind of mechanism to ensure the QoS in our system, therefore it is not an ideal choice for the time-sensitive application.
- **Differentiated Services:** Differentiated Services (DiffServ) are also known as soft QoS. In this model, it is allowed to differentiate packages among treatment categories. In the DiffServ there is no explicit resource allocation, therefore this could have better scalability than the IntServ model, and it has an easier implementation as well.
- **Integrated Services:** The Integrated Services (IntServ) model is also known as hard QoS. In the Integration Services model the applications have to ask the network for explicit resource reservation. This resource reservation will be guaranteed and predictable, therefore it would be great for real-time applications. However, this model has a huge disadvantage due to the fact of the limited resource of the system: if an application has a reservation of a resource we cannot guarantee that resource for another application, and the result is poor scalability.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Although this quality model is initially defined for the Internet, which plays an important role in a CPS/IoT System, we have to consider other parts of the system as well. To provide the required QoS factor in the whole system we need to ensure the QoS on each layer in our architecture. In order to do this, we have to identify the IoT system's component from QoS point of view, and the threats for each component.
As an overall picture from the system, we will use the QoS architecture from Ren Duan et al [98]. In their work, there are three different layers in an IoT ecosystem:

1. Perception Layer
2. Network Layer
3. Application and Service Layer

In the perception layer, the main units are the IoT devices, which are capable to measuring, and interacting with the physical world and sending the data to the network layer. One challenge in this layer is that mobile devices have limited resources such as power consumption and computational power. Another challenge is the diversity of mobile devices regarding the manufacturer and connectivity. There are some widely used communication protocols for these devices. However, if we want to use a gateway application, which is responsible to communicate with the devices, it has to be able to communicate with all the used communication protocols in the system via different endpoints.

The next layer is the Network layer, with the main function of consuming the huge amount of generated data from the perception layers and producing it to the Application and Service layer. This can be done via different kinds of communication networks, for example Local Area Network (LAN), Wireless LAN (wifi), Bluetooth, or Near Field Communication (NFC). There are some Quality factors to consider here as well, such as network delay, and bandwidth, or packet loss rate.

The Application and Service layer as it can be seen from its name has two subsystems. One subsystem is the Service, which is responsible for data management. The Service subsystem consumes data from the Network layer, and store it in a data centre. The Application subsystem provides the user interfaces and applications built on top of the collected data from the sensors. The sensor data is obtained from the data centre, from the Service layer. The Application Subsystem is the end-point User Interfaces for the users, with different algorithms, data aggregation, and real-time monitoring capacity in the background. The main challenges in this layer are ensuring the Quality for the different applications, such as service time, accuracy, or service delay.

In CPS/IoT system layers 2.1.1 we have to deal with many different technologies. Some of them have already defined QoS measures - for instance, the network layer, but others - mostly sensors from the perception layer - do not have any QoS.

CPS/IoT application specific characteristics

However, in order to perform a QoS analysis for IoT systems, it is not enough to identify the architecture's layers. We have to make a preliminary investigation of the specific

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

application characteristic in a CPS/IoT system too [99].

Heterogeneity and connectivity
In our work, the connectivity and heterogeneity are interpreted in perception, and network layers. These characteristics mean that we have to be able to handle the connection with heterogeneous hardware vendors, communication networks. For instance, in a factory, there are different PLC vendors, such as Siemens [100], Omron [101], or legacy PLC systems. For all these different systems we need to implement a fast-paced connection.

IoT applications, Data flow
In a CPS/IoT system, we have applications fed with sensor-generated data. Since the services are depended on the sensor data we have to guarantee the data and the information correctness, reliability. In a CPS/IoT system the data arrives typically immediately but in small batches (smaller than one Megabyte, usually from X*10 Kilobyte to X*100 Kilobyte), therefore the chosen database has to handle many small requests per second. The applications in the system contain the business logic such as the data aggregation and manipulation, for instance, filtering, joining, concatenation and they can be combined as a business process or workflow. There are different types of applications, such as Data analytic with machine learning models such a model for predictive maintenance, and real-time dashboard with charts and alerting systems. However, all of these applications must be able to handle the huge amount of data in real time.

Dynamic runtime changes
A CPS/IoT system can change dynamically, in a short time period. A change can be nearly anything, from the state of the device to the number of the devices, and many more. However, the changes not just happened with devices, but with the network, and services too. For instance, a network connection error can easily happen due to the legacy routers which operates with old standards and slower bandwidth in many factories. The network instability, poor mobile network, network delays, or network reliability are also challenging. From the service side, we need to implement a sufficient resource allocation and scheduling system to handle the changed resource consumption. For instance, if we operate in a factory, we are required to handle the connection of a new robot at any time, which mean change in every layer. A new production robot means hundreds of new sensors, with a huge amount of data. This sensor data has to be collected, stored, and handled such all the others in our system.

Safety
Our system design needs to contains appropriate security solutions. This means a variety of security consideration such as authentication, authorization or data encryption. We have to think about security in all layers, which means different techniques for every layer through the whole system. For instance, in the sensor layer, it is not enough to provide the software safety, as a CPS system, we also have to think about the physical attack as well. A physical attack cannot be just a deliberate event, it also can happen accidentally by an incidence, for example, a careless worker mutilates a sensor in a production machine. Nevertheless, the cyber-attacks in the network layer also happen frequently, therefore we also have to employ services to defend that layer as well.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.8.3 Characteristic - Quality attributes mapping

As a baseline we have chosen the ISO25010 quality model [102] which is used as an international standard for the evaluation of a software quality. Although ISO 25010 is originally used to measure a software quality, and a CPS/IoT system not just simply a software, ISO25010 can be used as a guideline due its broad quality attributes and hierarchical structure. However, in order to use this quality model, it has to be supplement with additional attributes from IoT architecture, for instance the well-known network QoS attributes.

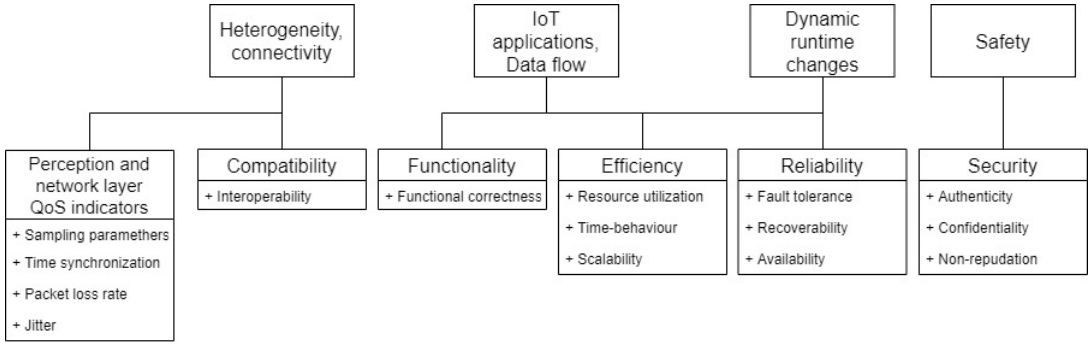


Figure 2.17: Customized quality model

As it can be seen in the Figure 2.17 all the CPS specific characteristics have the corresponding quality factor in the customized quality model. Similarly to the original model each quality attribute have sub-factors. In the following, we will define these quality factors and their sub-factors.

Heterogeneity and connectivity - Compatibility, Perception and network layer QoS indicators
Compatibility is the factor which identifies if the software and hardware components can work together seamlessly.

Compatibility subfactor:

Interoperability. Interoperability such as the compatibility, it is the factor how two or more system can operate and exchange information with each other and use this exchanged information. Interoperability is a very important factor for the CPS/IoT ecosystem where - as it was written before - there are many different vendors and systems.

Since heterogeneity and connectivity characteristic is mostly dealing with perception and network layer, we have to add their QoS indicators to the model.

Perception indicators:

Sampling parameters: Parameters for the specification of the sampling attributes from the device, for instance, frequency, precision.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Time synchronization: The devices have to keep uniform time reference, in order to be sampled correctly.

Network indicators:

The Efficiency factor's time behaviour subfactor includes most of the network indicators(since they are time-related), but there are other indicators too:
Packet loss rate: It indicates a percentage, of how many packets reaches the destination.

Jitter: Jitter is the variation of packet arrival time (latency). In real-time use-cases, we would need a consistent latency, which means jitter is as low as possible.

Bandwidth: The amount of data that can be transmitted in a fixed time period.

IoT applications, Data flow - Functionality, Efficiency
Functionality provides the degree to measure if the CPS/IoT system has all the needed software functions, and software - hardware connections to serve the given use-case.

Functionality sub-factors:

Functional Correctness: The Functional correctness shows if the software or hardware function produce the expected output for each input. To provide the functional correctness we have to define our requirements, and specifications clearly. The precise specification helps the developers to cover their functions with meaningful tests.

The Efficiency factor also important for the IoT applications and Data flow, it will be explained in the follows.

Dynamic runtime changes - Reliability, Efficiency

Efficiency is the metric, which measure the product operates at the same level of performance regarding the used resource and time.

Efficiency subfactors:

Resource utilization: For CPS/IoT system with dynamic changes - in the number of sensors or the resources - it is important to have a suitable resource utilization. Resource utilization means the system uses a suitable amount and type of resources to provide stable functionality. Utilization includes resource allocation, deallocation, scheduling, and management.

Time-behaviour: Time behaviour tests the system functions response time, throughput ratio, delay, processing time under given environmental conditions. In a CPS/IoT systems, it could be very critical factor depends on the use-case: in self-driving systems, the quick (real-time) response time must be contiguous, and predictable. The real-time behaviour importance is also true for the factories, where we want to know immediately if one of the machines has been out of order.

Scalability: Scalability is a measure, which indicates the capability of the software system to expand with new components - software and hardware components too. This is one of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the most important quality factors for many CPS/IoT use-cases, such systems in the factories, due it could have a great performance and response time for a few robots, but a production factory could comprise hundreds of lines with tens or hundreds of machines.

Reliability is the degree of the system, which measures the probability that the product will be failure free under specified conditions.

Reliability subfactors:

Fault tolerance: Fault tolerance is a degree of the system to show it is working properly during a failure. The failure could be a hardware or software component malfunction.

Recoverability: Recoverability is the capability of recovering from a fault occurred by a hardware or software malfunction.

Availability: Availability the degree of software, which shows, that the platform - including the hardware and software components - are available and operates according to the specification.

Safety - Security

A CPS/IoT system could handle much sensitive information - just think of the industry use-case: in a factory, many private data generated, for example, the setup of the robots, or more important: a new type of product. None of the producer companies wants to share their innovation or best practices with the concurrency.

Security sub-factors:

Authenticity: Authentication is the process when we control who can interact and use our system due to verification of the identity of a user or process.

Confidentiality: The measure of a CPS/IoT system, how can it protect and limits the access to the data and information from unauthorized access. Authorization is a process when the system checks the user rights of already authenticated users.

Non-repudiation: In the CPS/IoT systems there are such events, where it is important the prove that the event has taken place and cannot be repudiated later.

2.8.4 Metrics for quality attributes

As it can be seen we have assigned quality attributes for each characteristic of a CPS/IoT system, but it is not enough: we have to be able to measure these attributes in order to evaluate the system's performance. In this section, we will recommend a metric to each quality attribute.

Compatibility - Interoperability

The main metrics for interoperability is the quantity of all the possible connection endpoint what the system offers. These endpoints can be found in different layers of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

our system: in the perception layer, we have to be able to communicate with different kinds of units, for example in industrial use case these units are sensors, PLCs. In many factories, there are old legacy systems, for what we have to prepare our system too.

Perception layer indicators

For sampling parameters, we can define various metrics, as it was written before. The sampling frequency indicates, how many times we sample the unit in a period of time. Precision deals with the message delivery property: for instance in MQTT we can define, that our message has to delivered at least once. In that case, the service will send the message until the receiver acknowledges receipt of the message. For the time synchronization, the easiest metric is the count of devices, which are out of synchronization.

Network layer indicators

For the packet loss rate, we have to sum the packets, which reached the destination and divide them with all the sent packets. For the Jitter, we have to calculate the sum of difference between samples of latencies and divide them with (number of samples - 1). For instance, we have the following latencies: 2, 6, 3. The average latency is 3.6. The Jitter is calculated by taking the difference between samples: 2-6 = 4, 6-3 = 3, thus the total Jitter is 7/2 = 3.5. For the bandwidths, we have to divide the transmitted packets with time.

Functionality - Functional correctness

In functional correctness as we described before, we want to check if the output of the software function or a hardware unit is accurate to our expectations or not. One of the most essential technique for this is testing. The test scenario should be a very well documented progress, with separate steps and components involved. Testing the hardware components and the software functions is a little different, the main disparities between hardware and software test is the re-usability: we can use a software test easily with other parameters, but this is not true for the hardware test in the majority of the cases. An important type of test from our point of view is the unit test. The unit test focuses on the individual units of the hardware or software component. For example, the unit could be a sensor, and the test for it could be if it measures the distance accurately or not. In the software side, the unit could be the functionality which uses this measurement data, and the test, whatever or not it could show the data in the user interface correctly. Another important type is the integration test when we are testing if the system components can work together. For example, if the data collection application can connect to the sensor and collect its data.

Efficiency - Resource utilization

For measure, the system resource utilization the simplest metric is monitoring the system component resource requirement in a given time period – for instance, one minute. The resource requirements involve CPU, memory utilization, Ethernet throughput. This

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

metrics can be compared immediately which helps to identify possible bottlenecks in the system.

Efficiency - Time behaviour

Time behaviour metrics are about how the system deals with time. One of these metrics is the response time: a total amount of time needed for a system to respond to a request. For example, if we want to know the current temperature in a building the response time would be the time between our request sent to a sensor and the result of the current temperature. Another metrics is the throughput rate: the number of units (requests, sensor data, etc.) handled by the system in a given time period. Another one is the time delay: the time between two events. For example, how much later we can see the sensor data - for instance, the current temperature - in the user interface. These metrics play a very important role in the quality model, due to the use case possible time-sensitive behaviour as it was explained before.

Efficiency - Scalability

There are two types of scalability: First, if we add more resource to our system, how its performance grows. The second - which is probably more important for us - when the workload increases, how the resource utilization grows. With other words: the request rate of the number of connected devices over time. For example, when we add to the system one more production machine with hundreds of new sensors, how the new sensors increase our system resource utilization and response time. Also, it could be a good metric if we divide the response time by the used resource. Another possible measurement if we project the resource utilization to one sensor: how many resources needed to store and process one sensors data.

Reliability - Fault tolerance

One possible metrics for fault tolerance is the Mean Time Between Failures (MTBF). MTBF is defined as the number of hours the system operates, divided by the number of failures encountered. For example, if the system operated one week with three failures, then the MTBF would be $168/3 = 56$. Another metric for fault tolerance is Mean Time Taken to Repair (MTTR). MTTR is defined as the average time spent on repair the system and places it back to the operation. Another one is the Mean Time to Failure (MTTF) which describes the average amount of time until a failure occurrence.

Reliability - Recoverability

Metrics for Recoverability typically deals with the time needed to recover the system from failure. One of these metrics is the Recovery Time Objective (RTO) which shows exactly how many times needed to fix the system. For example, if the RTO is 30 minutes, that means if the system meets some crash, it has to be fixed in 30 minutes. Another metric is Recovery Point Objective (RPO) which is deals with the maximum amount of data what the system can lose. For example, if a failure occurs in the factory where the RPO is 30 minutes, then this means, it cannot be lost more than 30 minutes of sensor data.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Reliability - Availability

Availability can easily measure by divide the sum of downtime - when the system does not operate - with the total system operation time. There are high availability standards which start from 99.9 (three nines) when the total downtime of the system is 8.77 hours per year. The high availability can go all the way up 100 - which means no downtime, while the 99.995 (four and a half nines) means only 26.30 minutes downtime per year.

Safety - Authenticity

A good metric here is that if there is an authentication process or protocol in our system or not. For example, if we use an authentication system, which stores the user password in plain text format, it is better than nothing, but this system cannot be described as proper security. There are different ways to authenticate a user in a system: it could be done with the user name and password, an ID, or a session token, or with biometric identification (fingerprint, face, eyeball). Another good metric if the system contains any mechanism if the user failed in the authentication process X times – such as in the PIN code of the mobile phone: we have three chances to type our valid PIN numbers.

Safety - Confidentiality

One metrics for confidentiality could be the Access Control List (ACL). With ACL file and folder privileges could be defined, based on user account, or user groups. For example, in a factory, the worker in the production line cannot access to the raw sensor data, just the aggregated result, while the data analyst can access it. With ACL the privileges can be handled easily, with roles and groups: we can define distinct roles such as worker, or data analyst. A new employer or employee only has to be categorized based on his/her job description to be granted access to objects.

Safety - Non-repudiation

Metrics for non-repudiation is mostly about the system logging capabilities. For instance, if a worker opened a file – without permission -, and modified it, with an appropriate logger system we could check which file was corrupted, and when the event was. For non-repudiation we would need an identity (to check which worker did the event), the authentication of that identity (what profs the worker identity and the action of he or she logged into the system) and evidence which in our case a log entry with the exact timestamp of the event, and an event description.

2.8.5 Compare different use cases based on their quality models

In the work from Ren Duan et al [98], they divided the IoT applications into four distinct categories. From these four we will use three: I control, II real-time monitoring, III non real-time monitoring. In the following, we use these categories and present three CPS/IoT use-cases from different application areas, as an example of the usage of our quality model.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Control - Autonomous driving systems
Nowadays one of the most frequent control type IoT application is the autonomous driving systems. The goal of these systems if to achieve autonomous operation of a vehicle in the urban environment, with daily unexpected situations. To achieve this, IoT systems must operate in a deterministic, reliable fashion with strong security policies. For these applications the compatibility not so important since they will operate with the same devices during their lifetime: it is not a usual activity, to change control unit in a car. Efficiency is an important point during the development of autonomous driving. Of course, we can not consume as much electricity or computational resource as in an industrial use-case used in a factory, but it is not as critical aspect as in small battery powered devices, with very limited computational power as they used in the smart agriculture use-case. For these applications, the reliability, security, and functional correctness are the most important aspects, since man's life depends on them. The system has to provide high availability in every layer, in order to be able to react to every situation. As it was mentioned, these services are extremely delayed sensitive, therefore we have to ensure, that all the component is capable to handle real-time actions. As a control type IoT application, - where the IoT system controls other systems - it has to operate as an integrated service, with guaranteed resource reservation.

Real-time monitoring – Predictive maintenance
In the factories, there are many robots with IoT capabilities, which can be used to build a system for real-time monitoring of the production machines. This monitoring facility then can be the data source for the implementation of a predictive maintenance algorithm. Predictive maintenance is a process when the machine's sensor data – what is describes the machine current physically parameters - is analysed with the purpose of detecting the failure of the unit before it actually happens. Usually, this analysis is done in a huge historical data set from the machine, which may contain various patterns or observable trends about the malfunction. If we know with high certainty when will the machine fail, we can define a trigger event for the maintenance in the real-time monitoring system. With the predictive maintenance, we can save money and time, which is wasted for operations maintenance and production outage. In such use-cases, the compatibility is very important, since there are so many factory robot vendors, with different kinds of PLCs. In a factory, we also have to think about the legacy systems. Furthermore the reliability: in a factory where the production is going in three shifts, there is no place for unreliable IoT system. As it is mentioned before we have to pay attention to the security: an industrial system contains so many sensitive data which have to be protected from the external, and also from the internal attacks. These solutions save a lot of money and time for the production plants, but it is only possible if the system is functionally correct. As a real-time type IoT application, we have to handle it as a Differentiated service and treat it accordingly.

Non real-time monitoring - Smart Agriculture
In agriculture, the role of smart devices is emerging. Most of the agronomist use traditional methods, but with the growth of the population, the use of IoT systems

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

would be indispensable. To achieve an increase in production, different kinds of sensors would be needed to monitor the soil, temperature, rainfall, humidity, and other kinds of environmental variables. In the agricultural use case, the compatibility may matters since there are a variety of devices – just among the sensors - appear in one system. The reliability is one of the most important in the smart agricultural use case. The sensors have to deal with the extreme alternation of the weather conditions and they must operate 0-24. In addition, efficiency is as important as in the reliability, since the sensors use a battery for their energy consumption. However, security is not as important as in the other use cases, since the information flows in the system is not so sensitive: everybody can have the weather data, so it is not worth to hack the system. Finally, the functionality is as important as in the other use cases: these systems help to an agronomist to produce more goods, due to the application's capabilities. As a non real-time monitoring system, it can be categorized in the best effort QoS category.

In order to compare the importance of the quality models of different use cases, we defined importance weights for each quality attributes (see in Table 2.1). Let the sum of the weights equal to 1, in order to express the trade-off between the attributes. Than we can compute the overall value for quality model of an CPS/IoT application as follows:
in the explained quality modes, there are main quality attributes, with subfactors. The overall value of the *ith* quality attribute can be calculated:

$$Qa_i = \frac{(\sum_{n=1}^{n} sq_i)}{n} \tag{2.1}$$

Where Qa_i is the *ith* quality attribute, and sq_i the metric value of the *ith* subfactor. After this, we can calculate the overall quality model:
let w_{qi} the *ith* weight, and Qa_i the overall value of *ith* quality attribute. Since there are six main quality attribute in the system, we can define the overall value as follows:

$$Q_m = \sum_{n=1}^{6} w_{qi} \times Qa_i \tag{2.2}$$

Example weights for the explained use-cases:

Quality Attributes	Autonomus driving	Industrial IoT	Smart Agriculture
Perceptron and network layer indicators	0.14	0.1	0.2
Compatibility	0.08	0.15	0.08
Functional suitability	0.2	0.16	0.21
Efficiency	0.08	0.08	0.3
Reliability	0.3	0.26	0.15
Security	0.2	0.25	0.06

Table 2.1: Weights for Quality Attributes

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 3

State of the art

3.1 Related works

3.1.1 Deployment

In the study from Bellavista, Paolo and Zanni, Alessandro [103], they did experiments about fog deployment via containerization. They created a container for fog nodes, and tested their scalability, interoperability, and portability, on RaspberryPis. For the testing purposes, they deployed a Smart Connected Vehicles application.
While in the work from Antonio Brogi and Stefano Forti [104], they proposed a QoS-Aware deployment of IoT application through the fog. They specified a model, which can describe the intrinsic attributes of the system, and an algorithm, which can find a suitable deployment for the applications. They created a practical implementation from the algorithm, a Java tool, namely FogTorch.
The project Topology and Orchestration Specification for Cloud Applications (TOSCA) [105], is an OASIS standard, with the aim of improving the portability of applications in heterogeneous environments. It can be used for the description of the internal topology of the individual IoT service components, and the deployment procedure of IoT them.

3.1.2 Runtime Monitoring

According to the study from Rabiser et al, there are various existing monitoring approaches. They provide a comparison framework with four dimensions, namely context, user, content and validation. During the work, they identified a wide range of approaches, for instance, an SOA based implementation from Spanoudakis and Mahbub [106] or the work from Montali et al. [107], with industrial cooperation.
The work from Tharam Dillon et al [108] has the topic of Providing Quality of Service (QoS) in a Heterogeneous Systems-of-Systems Environment. They identified open challenges, for instance, latency issues with multi-protocol, sensor failure or system's

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

overall scalability. They also proposed a framework called Web-of-Things for CPS, which provides end-to-end QoS for all system components, including physical, cyber and communication.
The other work from Liviu Miclea and Teodora Sanislav [109], studied the dependability in Cyber-Physical-Systems. After the introduction of the theoretical background, they used a Hydropower system as a case study. They had serious requirements, and constraints with this system, such as availability, and correctness, self-test capability, data security and reliability. As a result, they created a system model, where interdependencies in CPS can be expressed with measurable attributes.
The most similar work is the study from Radu Boncea and Ioan Bacivarov [110]. They compared various monitoring systems such as Prometheus, InfluxDB, OpenTSDB, and Graphite. As a result, they proposed an automatic monitoring architecture, based on Prometheus, message broker, and Reactive applications.

3.1.3 Quality of Service analysis

In order to provide an appropriate QoS in CPS/IoT system, we have to ensure a certain QoS level in every connected layer, systems, and services, therefore there are many different kinds of research studies in the theme [111].

Some studies focus on the QoS in IEEE 802.11 wireless local area networking (WLAN) [112] standard [113]. This is a very important aspect since in the CPS/IoT system operates mostly in a wireless network environment. The main focus in this research topic was on physical and Media Access Control (MAC) layer. In early 802.11 standards (802.11-1997 [114]) there is no explicit mechanism for service differentiation, therefore it is difficult to grant sufficient QoS level. In the following standards - 802.11 a/b/g/n [115] [116] [117] [118] - there is some improvements with the support of the application differentiation, and in data rate. However, they have also various limitations, such as they do not incorporate admission control, which can occur performance problems. In the newest standards - 802.11 n/ac/ad [118] [119] [120] - they used a scheduling mechanism, and admission control algorithms, therefore these standards support different application QoS requirements.

Other studies [121] is about more specific communication protocols such as IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) [122]. 6LoWPAN is a standard, which allows the deployment of a vast number of smart objects in a local wireless sensor network. However, 6LoWPAN from the security point of view is not the strongest protocol since the used cryptography technologies can be broken via the weak LoWPAN devices and wireless environment. This weakness has QoS-related security threats, thus we have to consider during the planning of an IoT system.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

3.1.4 Arrowhead

Quality of service in Arrowhead framework

In Arrowhead there is a concept for supporting the QoS within the framework [123]. They focus on four QoS properties:

- Delay: Delay in Arrowhead refers to the execution of communication within a time frame. Arrowhead defines hard real-time and soft real-time constraints for the deadline.
- Bandwidth: Bandwidth refers to the fact, that in the system there is a minimum transfer rate guaranteed for the data produced in a time unit.
- Resources Limits: Resource Limits protect the architecture by limiting the resources that the system or service can consume
- Communication Semantics: Communication Semantics assure that the message is received at least once, without duplication in the same order as it was produced.

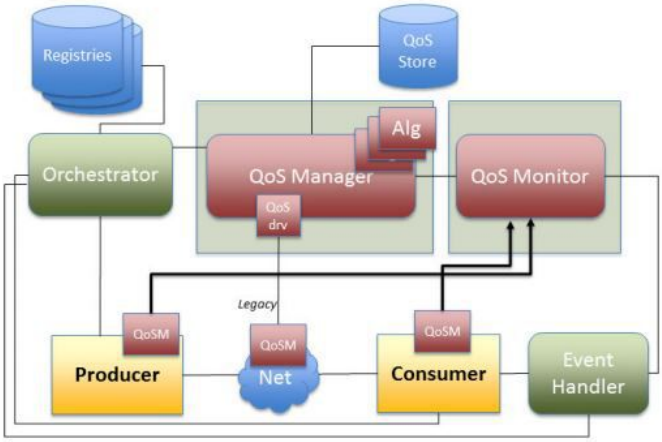


Figure 3.1: Architecture for QoS in Arrowhead

In order to ensure the QoS within the system, they have implemented two Systems, namely: QoSManager, and QoSMonitor.

The QoSManager is a plugin for the Orchestrator service with the role to verify the QoS requirements of a new Service. When a new service is added to the system, it can have various QoS requirements such as traffic priority, delivery guarantees of the message, etc. During the registration of a new service, the Orchestrator requests the QoSManager to perform reservation in order to grant these required QoS properties.
The QoSMonitor receives a set of characteristic to monitor from the QoSManager. It

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

consumes the services for this information. The main function of the QoSMonitor is to implement real-time monitoring for the performance of services, system and devices, and check the violations of Service Level Agreement (SLA). If the service or a system component cannot meet the pre-defined QoS, QoSMonitor informs and alerts the corresponding service about the event.

Arrowhead QoS algorithm

In Arrowhead each application has a fixed priority and the activities of an application are scheduled based on this fixed priority. The priority of an application inversely proportional to its deadline. Each activity has a time frame to complete. To verify the QoS, the time of the activities has to be summed for each application and compare them to the application deadline. If the total activities time is smaller than the deadline the QoS is feasible.
According to the System design document of the last stable version of Arrowhead Framework, "the QoS Manager concept is yet restricted to only handle communicational QoS" [124].

3.2 Comparison and summary

As a comparison, we can see, there are similar project in deployment, and monitoring as well, such as [103], where they used the same containerized deployment approach with Raspberry Pis, or the work from [110], where they used the same Prometheus monitoring solution as in our work. However, there are different approaches as well, such as WoT, or ForTorch, where they implemented the whole system from zero. In the comparison with the current state of the Arrowhead project, we can see many differences. From the deployment aspect, Arrowhead uses manual build and deployment methods at the moment, and this work is part of an a ongoing activity to ensure fully automated CI/CD. From the monitoring, and QoS point of view, the main difference between our work and the Arrowhead QoS concept is the ability of doing corrective actions in order to recover the optimal QoS within the system. This means that our QoS Monitoring and Reconfiguration Service will not only send an alert in case of any QoS violation, but it will perform an action as well. The proposed QoS services in Arrowhead are in a conceptual phase and they are not included in the latest official release. Our work will introduce new segment of connecting QoS with continuous integration and monitoring. Providing ability to the system to reconfiguration, redeployment and adaptation to emerging changes in an execution environment makes CPS/IoT system more robust.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 4

Methodology

For the first step of the work, we are going to perform a comprehensive analysis and literature research about the various QoS properties of IoT systems. The outcome of this analysis is a custom quality model. This quality model supports the comparison of different CPS/IoT use-cases from various application domains.

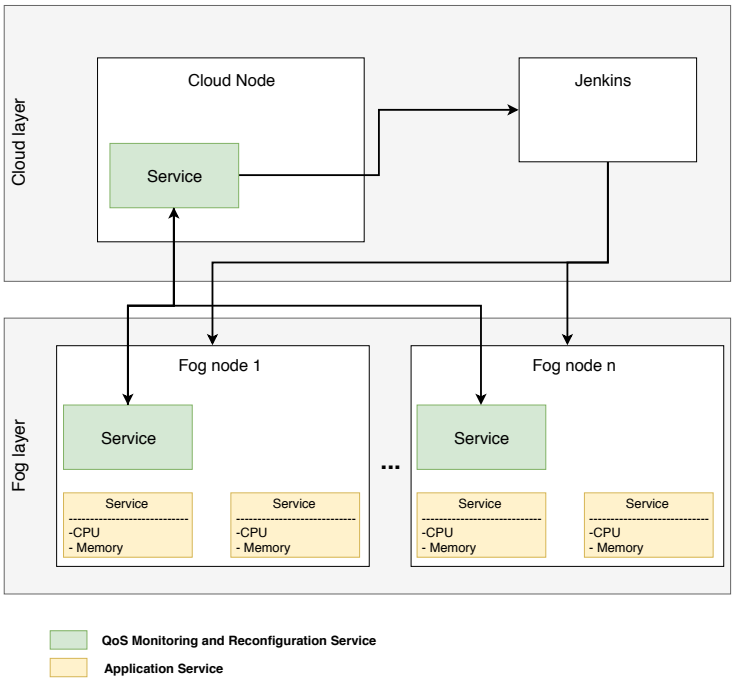


Figure 4.1: QoS Monitoring and Reconfiguration Service

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

As a first step of the practical implementation, we are going to establish an experimental environment, where we can develop and test different deployment and monitoring approaches. For this environment, we need to install a hardware infrastructure, which follows the Fog computing principles. The system includes various hardware devices in order to simulate real-world architecture more accurately. For the sensor layer, we will use a simulation program on a PC with a Windows operating system. In the fog layer, there will be multiple Raspberry Pis [125]. This is a relatively reasonable device to build a system with multiple nodes. In the cloud layer, as a one node cloud cluster, another PC with a Linux operating system is going to be used.

The software stack of the experimental environment has four main components:

The first component of the system is a service framework based on the Service Oriented Architecture. Each service within the architecture provides different functionalities. Some services are essential for continuous operation of the system, therefore we need to guarantee the optimal working conditions and requirements for these services. In order to overcome the problems with the heterogeneous hardware, and software structure, we are going to employ a container-based approach. As it was mentioned before, in a container we encompass the service with all of its dependencies, and runtime environment. Therefore with the container-based approach, we have the ability, to execute the services in any suitable node within the cluster, without any additional installation or configuration.

The second component is the Continuous Integration and Deployment system. This system is responsible for the testing, validation, and deployment of a new application or a new version of an existing service. In CI/CD systems, we can specify a pipeline, with multiple stages. Our pipeline implementation will contain four stages: pull the latest source code of the service from a version control system, test it, build a container image from it, and deploy it to the cluster. In case of any failure, such as a failed test, the pipeline gets interrupted, which prevents the deployment of service to the production environment.

The third component is the monitoring architecture. The monitoring system can observe all the individual services, and nodes, and in case of any unexpected system behavior it performs an alert to a pre-configured endpoint. In order to define the optimal system behavior, the monitoring architecture has to support system specification in formal languages. Since a CPS/IoT architecture contains hardware with low computational power, the monitoring system has to operate with the minimum possible resource requirements.

The fourth component is the QoS Monitoring and Reconfiguration service. This application receives alerts from the monitoring system and analyses the possible corrective actions based on its application domain knowledge and predefined problem resolutions. However, the decision on the solution is not trivial: for instance, if a node hosts multiple services, and they interfere with each other, we have to analyze, whether one of them can be deployed into another node or not. Another possibility, if a service experiences a failure, we have to check if it has a previous version available, or we can migrate it to

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

another suitable node. However, if we can not do anything else to restore the system optimal behavior, we must bring it into a safe state, where it does not function, but it does not harm its environment. This is very important in safety-critical use-cases, such as industrial IoT.

After the implementation, we make experiments with the QoS Monitoring and Reconfiguration service, which can be seen in the Figure 4.1. For this testing purposes, we are going to create an experiment environment with services, which artificially create QoS violations, such as increasing resource (memory, CPU) consumption. During the experiment, we validate the behavior of the system and compared the solution with a pre-defined configuration. The experiments include scenarios for each possible corrective actions. We are going to have a real-world use case also, where we will implement an industrial production environment. This includes a simulation of a conveyor belt, multiple services for the data collection and storage, and a User Interface with a real-time dashboard and simulation controlling ability.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Implementation

This chapter focuses on the exact implementation of our proposed architecture, and our contribution to providing deployment, monitoring and certain QoS in CPS/IoT systems. All the source code for the implementation can be found in a GitHub repository: `https://github.com/tuw-cpsg/smartproduction`. An overview of the implementation can be seen in the Figure 5.1

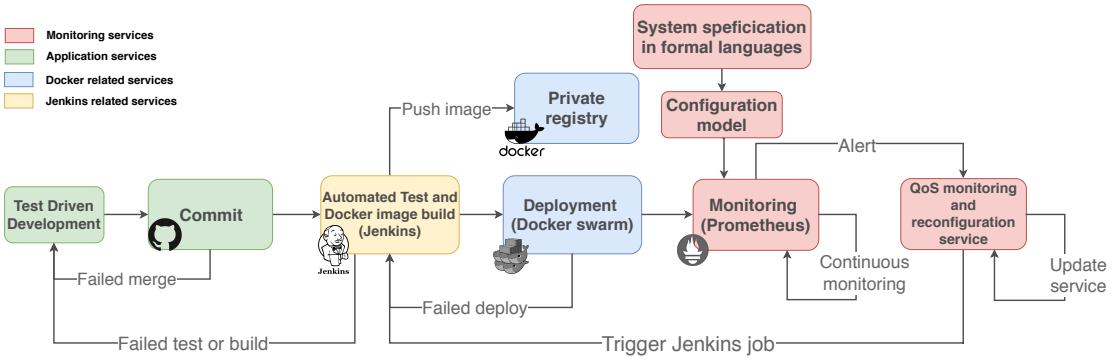


Figure 5.1: Implementation overview

5.1 Hardware infrastructure

For the demonstration of our work we constructed a lab experimental environment in three layers of operation. As it was described in the 2.1.1 section, this architecture has three distinct layers. In the first - device layer - we had a PC with a Windows operating system on it. This PC runs the chosen simulation program, which will be explained in the 7.1 section in more detailed. In the fog layer, we had two Raspberry Pi 3 Model

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

B [126]. This is a relatively cheap alternative to build a system with multiple nodes. Although it is cheap, it has a really decent hardware configuration, which makes it a perfect candidate as a fog node:

- Quad-Core 1.2GHz Broadcom BCM2837 64bit CPU,
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet

In the cloud layer, we had another PC with Linux operating system on it. In our hardware environment we only have one cloud node, however, it is possible to use multiple of them. The cloud node's hardware configuration:

- Debian GNU/Linux 9.4
- Intel(R) Core(TM) i5-7600 CPU @ 3.50GHz
- 16 GB RAM

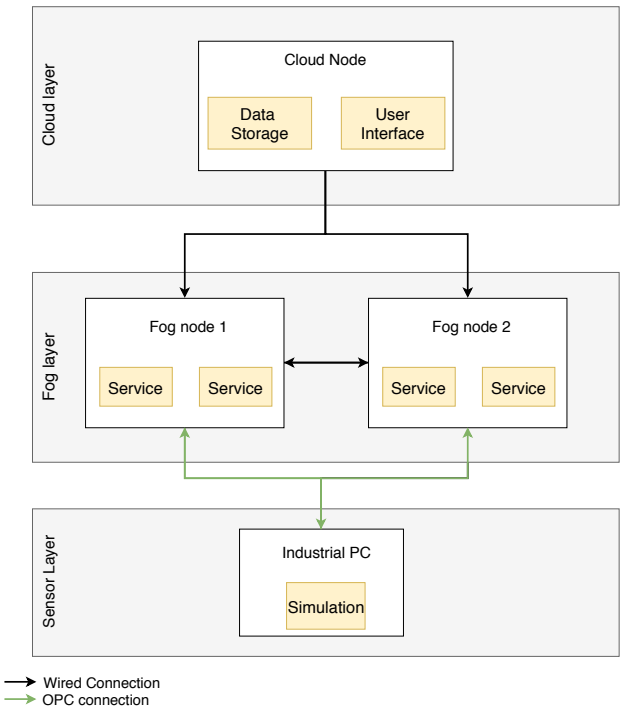


Figure 5.2: Hardware architecture

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

After the hardware description of the hardware infrastructure, we describe our software architecture.

5.2 Docker implementation

In order to used contenarized deployment, and runtime environment, we had to containarize all used services, including the Monitoring architecture, Deployment architecture and QoS Experiments. During the work we used Docker. All the Docker-related code can be found at `https://github.com/tuw-cpsg/smartproduction/tree/master/docker` .

As it was mentioned in the section 2.5.1, Docker has a cluster management facility called swarm. In order to use Swarm, we had to initialize it within our infrastructure. The first step of the initialization is to run the following command in the node, which we want to be a manager. In our case, this node is only one cloud node.
`docker swarm init –advertise-addr <cloud node IP address>`
After that, we can add nodes to the cluster with this command:
`docker swarm join –token <Swarm cluster token>`
`<Manager node IP address>`
As can be seen in the join command, there is a token, which is unique for each cluster. The `swarm init` command generates two random tokens, one for the workers, one for the managers, which can be used to connect new nodes with a worker or manager role accordingly. In the Figure 5.3 can be seen our Swarm architecture.

Figure 5.3: CPS/IoT Ecosystem Docker Swarm Architecture

Since the Swarm cluster consists of multiple nodes, we have to distribute the Docker images to all of them. For this, there is two way: one is to use official Docker Hub, the other is to deploy an own image registry. Considering, that this is an experimental development, we wanted to use our own image registry. Image registry is a version-control system for Docker, where we can store images with different tags, for instance, `latest`, which refers to the latest version of the image, or `arm` to indicate the used architecture. During the deployment of the image, we can specify this tag, in order to use the right version.

The local image registry can be deployed with this command:
`docker run –name imageRegistry –d –p 5000:5000 registry:2`
We also deployed a frontend service for the registry:

```
sudo docker run \
  –d \
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
--name imageRegistry \
-e ENV_DOCKER_REGISTRY_HOST=128.130.39.67 \
-e ENV_DOCKER_REGISTRY_PORT=5000 \
-p 8083:80 \
konradkleine/docker-registry-frontend:v2
```

In order to specify in each node's Docker engine, which registry to use, we have to modify their configuration:

```
sudo nano /etc/docker/daemon.json
{"insecure-registries" : [ "128.130.39.67:5000" ]}
sudo systemctl restart docker
```

After the initialization of the Swarm, and image repository, we give an outline about the Docker-related problems with the used services, and our suggested solutions for them.

As it will be described in the 5.3 section, we apply Jenkins as a Continuous Integration tool. For the deployment to the Docker, we used the official image of Jenkins. However, as a part of the Jenkins job, there is a building of new Docker images from source code, such as Arrowhead client services. For the building, we need the Docker daemon, but that is not accessible inside the container. As a solution, we mounted the host machine's Docker daemon in the container at the run:

```
docker run -p 8082:8080 \
  -v /var/run/docker.sock:/var/run/docker.sock \
  --name jenkins \
  jenkins/jenkins:lts
```

With this solution, Jenkins is able to use the host's Docker daemon to build images. However, we still need to install Docker binaries inside the container. Since the official Jenkins image is based in Debian 9, we can do the installation in such a way:

```
docker exec -it -u root jenkins bash

apt-get update && \
apt-get -y install apt-transport-https \
     ca-certificates \
     curl \
     gnupg2 \
     software-properties-common && \
curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg >
 /tmp/dkey; apt-key add /tmp/dkey && \
add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
   $(lsb_release -cs) \
   stable" && \
apt-get update && \
apt-get -y install docker-ce
```

For the purpose of using Arrowhead Framework with Docker swarm, we had to create custom Docker images for all the Arrowhead framework services. Since the Arrowhead services will operate on the fog nodes, - in our case Raspberrys - therefore we had to use Arm base images. The base image can be seen as an operating system, which you can

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

install or add software to. We used `arm32v7/openjdk:8-jre` as a base image, since all of the Arrowhead services are implemented, and complied with Java.

We can deploy the stack to the swarm, based on a compose file. The compose file is a Docker configuration file, where we can define a multi-container stack. Ours compose file for the industrial use-case can be found at [127]. In the industrial use-case, we have the base Arrowhead services and three custom Arrowhead client services. In this file, we can specify, beside others, the desired docker image, constraints for placement and resources, and even dependencies between services with `depends_on` argument. However this only guarantees, to start service's containers in dependency order, so the dependent service will be deployed secondly, but it does not keep track of the service internal state. This means, that Docker starts the containers in the right order, but if the startup for the service in the main container is longer than for the dependent container, the dependent service will probably fail. For instance in the industrial use-case, there is an SQL database for Arrowhead, and all the other services use that. However the startup for the database is longer than the startup for Arrowhead services, therefore the service containers failed. We solved this problem with `wait_for_it` script [128]. This is a pure bash script with no external dependencies, which test the availability of a service in the configured host and port. This script forbid the startup of the service inside the container, until the main service has successfully started. With this method, we prevent the failure of the service and with it the container too.

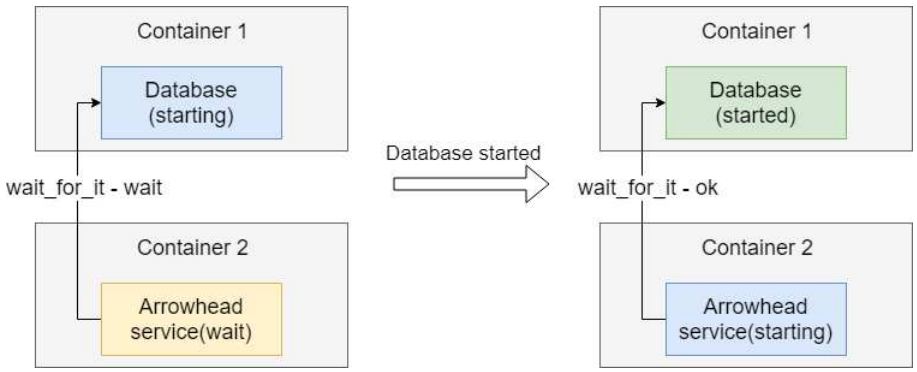


Figure 5.4: Wait for it script in usage

We had another difficulty during the deployment of the Arrowhead stack. The issue was about the web server address of the subscriber type Arrowhead client services.
Normally the subscriber service registers itself into the Event Handler, in order to subscribe to the events with its web server address. In Docker Swarm, this is not a suitable form, since we can deploy the service into any suitable node, therefore instead of specify an exact IP address, we configured the web server address as localhost. In Docker, we can specify the port mapping, which ensures the given port will be available in the host machine as well. For instance, if we start our subscriber service such as `docker run -p 8446:8446 ArrowheadSubscriber`, than this will map the container's localhost

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

8446 port to the host machines 8446 port, therefore the service will be reachable from outside. As can be seen, the localhost option always means the node, where the service is deployed. However, if the service registers with localhost into the Event Handler, then it can not find the service, and perform the notification. To solve this problem, we added one more configuration parameter, namely `dockerHost`. This parameter specifies the domain name of the service inside stack, which is discoverable inside a swarm stack network.

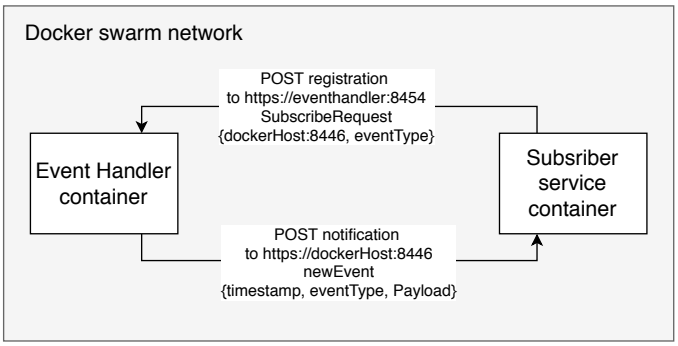


Figure 5.5: Arrowhead subscriber service in Docker swarm

5.3 Deployment workflow

For the deployment of Docker swarm stack into the cluster, we use Jenkins automation server in the CPS/IoT Ecosystem. Our process has six steps (as it can be seen in the Figure 5.6):

1. Jenkins got a trigger to deploy a job:
   Trigger from a version control push
   Trigger from the monitoring service
2. Jenkins build the docker images
3. Jenkins push the built images into a local Docker Registry
4. Jenkins deploy the swarm stack in the Docker swarm node
5. All the nodes pulls the latest images, in order to ensure, all of them use the same, and up-to-date image
6. Docker swarm orchestrate the stack

The trigger for the deployment can come from a version-control system, such as git, or svn as a reaction to an event, for instance, a push. Another source of a trigger is the QoS monitoring and reconfiguration service, it will be described in section 5.5.1. The trigger starts a preconfigured Jenkins job. In this Jenkins job, we pull the latest code from the version-control system and build a Docker image from it. After the successful build, we push the Docker image to the local image registry. As a next step Jenkins deploys the stack to the swarm based on the compose file.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

After Jenkins deployed the stack into the manager node, Swarm orchestrates it within the cluster. All the nodes pull the latest image from the local registry. After the containers start, Swarm maintains the cluster desired state, scheduling services, and orchestrating the tasks.

For the deployment purposes, we used Jenkins pipelines. Pipelines is a way, how continuous integration job can be defined. The pipeline is written in a text file, called Jenkinsfile, with Pipeline's domain-specific language syntax. With this syntax we can define our job with a declarative way, for instance a docker image build:

Algorithm 5.1: Jenkins Docker Image build

```
stage( "Dockerize Registry SQL" ){
  //define the agent, where we want to build the image
  agent{ label "build"}
  steps{
        //get the latest source code
        sh "git pull"
          script {
                //define local image registry
                docker.withRegistry("https://128.130.39.67:5000"){
                //build service registry image with tag of the BUILD_NUMBER,
          //from the $WORKSPACE/serviceregistry_sql folder
                customImage = docker.build(
                  "arrowhead-service-registry_sql:${env.BUILD_NUMBER}",\
                  "$WORKSPACE/serviceregistry_sql")
                customImage.push()}}}}
```

The biggest advantage of Jenkins pipelines, that we can put the Jenkinsfile beside the code in the repository, therefore we can track it more easily.
As a summary for the deployment, our workflow can be seen in the Figure 5.6.

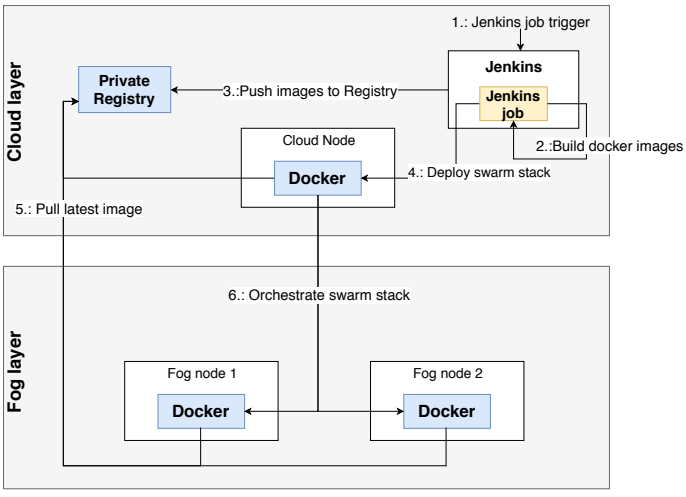


Figure 5.6: Deployment workflow in CPS/IoT Ecosystem

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.4 Monitoring architecture

After the presentation of the STL, and Prometheus in the 2.7 section, in the following we will describe the integration of these applications into the CPS/IoT Ecosystem. In order to use the Prometheus stack in a heterogeneous IoT environment, we had to implement a Docker swarm stack the monitoring services. This swarm stack involves the Prometheus server itself, an Alert manager, and the exporters. Since we have already used Grafana for other purposes, therefore we omit that from this stack. Our stack implementation can be found in [129], it is based on the this work [130]. During the specification of this stack, we had some constraint: the Prometheus server and the Alert manager have to run on the cloud node, and exporters have to run on both cloud and fog nodes. The first constraint can be satisfied via docker placement rules. The second one is required a bit more work since the cloud and fog nodes run on a different architecture (as it was described before). Docker solves this problem with a manifest list: this file represents all the architecture and platform which is supported by the specific Docker image. During the deployment the Docker engine automatically pulls the right version from the container, to grant a solution for the problem. Additionally, we configured the exporters as a global swarm service, which means swarm automatically deploys them on a new node when we add one to our cluster. This is very convenient since there is no additional configuration or deployment at a new node: swarm deploys the exporter services, and Prometheus discovers them via Service Discovery. The monitoring architecture in CPS/IoT Ecosystem can be seen in the Figure 5.7 below:

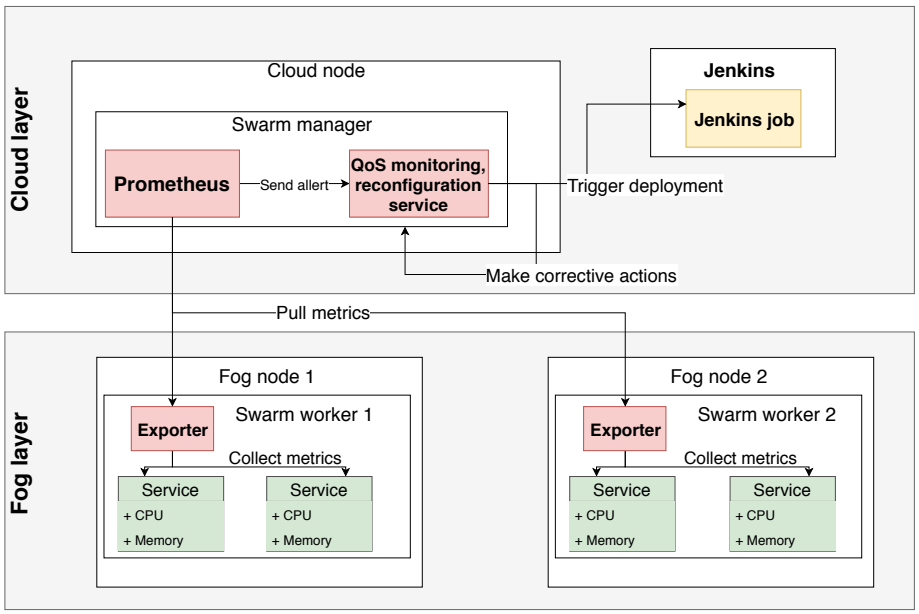


Figure 5.7: Monitoring architecture in CPS/IoT Ecosystem

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.5 QoS reconfiguration service

In our system, the monitoring is based on Prometheus. As it was described before, we have to use Node exporter, and Container exporter to collect metrics from nodes, and containers as well. In order to ensure a certain level of QoS within the system, we define a set of alerting rules. This rules can contain limitations regarding resource consumption, and service status. Prometheus server periodically validates these rules, and it sends an alert in case of any violation. The Alert manager receives these alerts, and decide where to send them towards based on a routing configuration. For example, if a container consumes too much memory, it will generate an alert in the Prometheus server. The server will send it to the Alert manager, which will perform an e-mail alert or a POST request on a web server. In order to make corrective actions, we implemented the QoS monitoring and reconfiguration service as an alerting endpoint. This application will be responsible to make corrective actions, based on alerting information, and application domain knowledge. The alerting messages contain various information among others about the alerting status, container name (if it is an alert for a container), the alert name, alert severity, a timestamp, etc.

5.5.1 Implementation

In this section, we will provide detailed information about the QoS monitoring and reconfiguration service. This is our custom developed application for the IoT Monitoring. This service is responsible for handling the incoming alerts and decide on the possible corrective action. The implementation can be found at `https://github.com/tuw-cpsg/smartproduction/tree/master/DynamicMonitoring`.

In the following, we will explain each step of the Algorithm 5.2. The QoS monitoring and reconfiguration service works based on a configuration file. An example configuration file can be seen in 17. This file contains all the necessary domain knowledge of the applications, which is required to decide on the possible corrective actions.
The first step of the algorithm is to check if the service ID exists in the configuration file or not. If there is no information about the alerted application than our service sends an alert to a designated developer.
In the second step, we obtain the list of possible corrective actions from the configuration. We can define multiple solutions if the service gets in the wrong state. For the solutions, we can define weights, which will determine the execution order. If the solution list is not empty, we sort the solutions based on these weights, otherwise, if the list is empty, or contains invalid solutions, we notify a developer. The current implementation supports four solutions: `memory reduction`, `CPU reduction`, `migration`, `rollback`. If the solution list is not empty and contains at least one valid solution for the service then we iterate through the list. For each elements, we have to check if the solution is in the currently supported solution list with `CheckSolutionIsValid(s)` function. If the solution is valid, then we have to check, if it is possible with the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

`CheckSolutionIfPossible(s)` function.

Algorithm 5.2: QoS monitoring and reconfiguration service

Input: Service ID *id*

```
1  if Service ID is in configuration then
2      Solutions := GetSolutionForService(id)
3      if solution exist for Service ID then
4          SortSolutionByWeight(Solutions)
5          for s ∈ Solutions do
6              CheckSolutionIsValid(s)
7              CheckSolutionIfPossible(s)
8              if possible then
9                  perform the solution
10                 break for;
11             end
12             else
13                 Iterate to the next solution
14             end
15         end
16     end
17 end
```

An example configuration:

```
{"services":{ //list of services
        "test_service_1":{ //service name
                "requirements":{
                "minimum_memory": 100000000,
                "maximum_memory": 120000000,
                "minimum_cpu": 5,
                "maximum_cpu": 70,
                "can_updated_alone": "True"
        },
        "solution_priority":{
                "memory_reduction":5,
                "migration":4,
                "rollback":3,
                "cpu_reduction":2,
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
                "notify":1,}
        },
        "test_service_2":{
                "requirements": {
                        "can_updated_alone": "False",
                        "jenkins_job_name": "resource_tester"
                        },
                "solution_priority": {
                        "migration": 2,
                        "notify": 1}
                }
        },

        "notify_e_mail:": "john.doe@corporation.com"
}
```

Listing 5.1: QoS monitoring and reconfiguration service configuration

The `CheckSolutionIsPossible(s)` function is individual for each solutions:

`CheckSolutionIsPossible(memory_reduction)`:
During the memory reduction, we want to limit the container memory usage. The developers can define hard memory limits for the application in the configuration file. Hard memory limits mean, we can not violate them in any circumstances. In memory reduction case, we have to be sure that the adjusted memory constraint (the new limit) for a service will be within the limits. If there is no minimum memory requirement defined, we consider it as 10MB. If the solution is possible, we adjust the container constraint and update the service.

The CPU reduction works similarly as the memory reduction, therefore we will not explain it in detail.

`CheckSolutionIsPossible(migration)`:
During the migration, we want to place the service into another node in the cluster. In this solution, we have to investigate, if there is another suitable node in our cluster regarding constraints. These constraints can refer to service placement, such as the role of the node, or it can be constraints regarding to the used architecture, or resource constraints such as minimum available memory in the node. However, there is another condition here: if we have multiple services (a service stack) we can decide on, if one service of this stack can be migrated alone or not. This condition also can be specified in the configuration file with `can_updated_alone` element. If this `False`, than it means, that we have to deploy all the services - the stack - again together. In this case, there must be a configuration element `"jenkins_job_name"`, which defines the name of the Jenkins job to trigger. If this requirement is `True`, which means the service can

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

be migrated alone, and there is another suitable node for within the cluster, then we add the current node Id as a permitted node, and we update the service.

`CheckSolutionIsPossible(rollback)`:
During the rollback, we want to deploy the container, with a previous image version. In this case, we check the image repository, if there is either a previous version for this container or not. If there is, we update the service with the last but one version of the image.

Any other case, when we can not find a suitable solution, or there is no other solution in the solution list or any other problem, we send an alert to a developer.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

Experiments

In this section we validate our solution with a set of experiments. During these experiments, we simulate the most common system faults. There are many kinds of errors, in a CPS/IoT system where we use heterogeneous softwares. One possible failure is when a service start consuming enlarged resources, such as CPU, or memory. Another case is when a service does not function well after an update. There are several causes for these errors, such as change of programming language, bad programming, lack of service testing, or interfere between services. We use QoS monitoring and reconfiguration service to detect and react in case of any system fault.

6.1 Service start leaking memory

Introduction:
This experiment demonstrates how QoS service detects and limits overhead memory consumption in a service container. In this case, the problem is the services memory leakage. We measure and limit the service memory consumption on the level of container.

Procedure:
For this experiment, we created a custom service with name of `resource_tester`, which was a python web server. It starts leaking memory when it gets a POST request. The implementation can be found at: `https://github.com/tuw-cpsg/smartproduction/tree/master/docker/images/Experiments/mem_test`.
The algorithm for the leaking service:

The Prometheus alert config:

```
- alert: test_service_high_memory
  expr: container_memory_usage_bytes{name=~"resource_tester.*"} > 120000000
  for: 5s
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Algorithm 6.1: Memory leaking service

Input: POST requests http://<serviceIP>:5555/start

```
1 while true do
2 |   add 5MB of space character to a string
3 end
```

```
  labels:
        severity: warning
  annotations:
        summary: "Test service high memory usage"
```

QoS monitoring and reconfiguration service configuration:

```
{"services":{
          "resource_tester":{
                    "requirements":{
                    "minimum_memory":  70000000,
                    "maximum_memory": 160000000,
                    "can_updated_alone": "True"
                    },
          "solution_priority":{
                    "memory_reduction":3,
                    "migration":2,
                    "notify":1,}
          }
}
```

When we sent a request to the `resource_tester` service it started leaking the memory. When it reached the limit, the QoS monitoring and reconfiguration service got an alert. As you can see in the Figure 6.1, the memory consumption is peaked at 170MB, then the monitoring tool checked the pre-configured possible solutions. For this experiment, we configured the memory reduction first. After the identified possible solution, the tool checked, if the memory reduction is possible: we can not set the lower memory limit to service, than the minimum limit given by the developer. Since it was possible (the minimum memory limit is 70MB, and the possible new limit would be 100MB), the tool updated the service with the new constraint.
It is possible, that the application can have enough memory after the limitation (if the developer has not set appropriate limit) and fail with Out Of Memory failure. More information about this can be found on the Docker website: OOM `https://docs.docker.com/config/containers/resource_constraints/`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

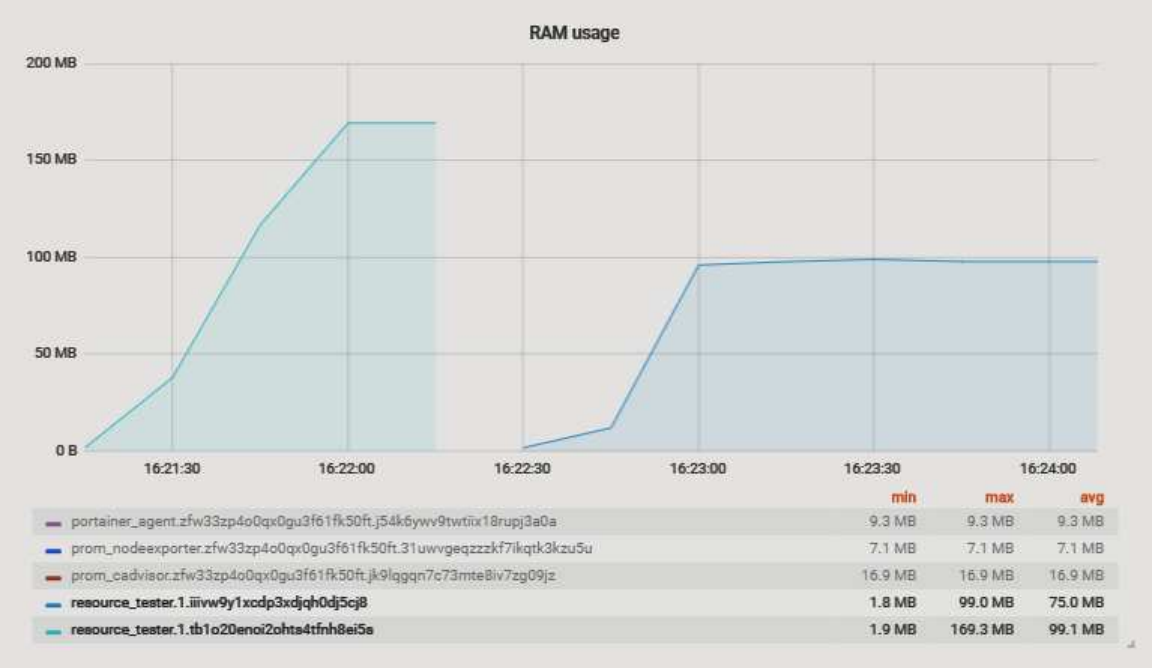


Figure 6.1: Service start leaking memory

The log from QoS monitoring and reconfiguration service:

```
2019-03-06 16:22:14,528 - INFO - identified order: [u'memory_reduction', u'migration', u'notify']
2019-03-06 16:22:14,528 - INFO - possible solution: memory_reduction
2019-03-06 16:22:14,529 - INFO - Try solution memory
2019-03-06 16:22:14,537 - INFO - Checking the limits
2019-03-06 16:22:14,537 - INFO - Current memory limit in bytes  160000000
2019-03-06 16:22:14,537 - INFO - The possible new limit would be: 100000000
2019-03-06 16:22:14,537 - INFO - The minimum memory requirement: 70000000
```

6.2 Service with increased CPU usage

Introduction:
This test is similar to the *Service start leaking memory*, but we simulates an enlarged CPU consumption of a service. In this case we also measure and limit the service CPU consumption on the level of container.

Procedure:
For this experiment, we created a custom service with Linux stress [131]. Linux stress is a tool to "impose a load on and stress test systems".
The implementation can be found at [132].

The Prometheus alert config:

```
- alert: test\_service\_high\_CPU
  expr:
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
sum(rate(container_cpu_usage_seconds_total{name=~"resource\_tester.*"}[1m]))
                / count(node_cpu_seconds_total{mode=system}) * 100 > 50
for: 5s
labels:
      severity: warning
annotations:
      summary: "Test service high CPU usage"
```

QoS monitoring and reconfiguration service configuration:

```
{"services":{
        "cpu_test":{
                "requirements":{
                        "minimum_CPU": 10,
                        "maximum_CPU": 50,
                        "can_updated_alone": "True"
                },
                "solution_priority":{
                        "memory_reduction":2,
                        "cpu_reduction":3,
                        "notify":1,}
                }
}
```

As the service started, the stress application allocated half of the available CPU. After the reconfiguration tool receives the alert from Prometheus, it updated the service CPU constraint to 25% of maximum memory usage. In the Figure 6.2 can be seen the total CPU usage of the cloud node (with name `groot`).

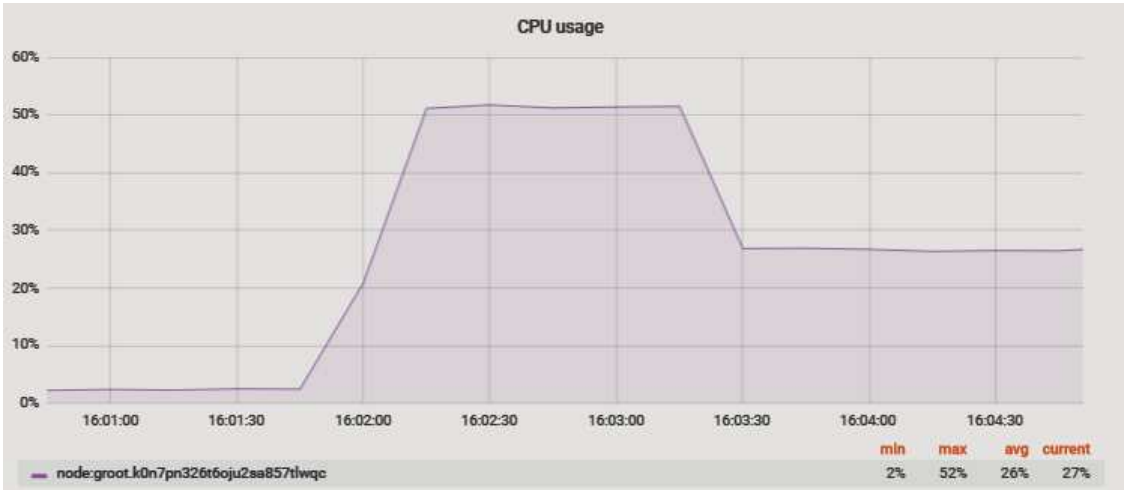


Figure 6.2: Service with increased CPU usage

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The log from QoS monitoring and reconfiguration service:

```
2019-03-07 16:03:02,126 - INFO - identified order: [u'cpu_reduction', u'notify']
2019-03-07 16:03:02,126 - INFO - possible solution: cpu_reduction
2019-03-07 16:03:02,126 - INFO - Try solution CPU
2019-03-07 16:03:02,127 - INFO - Try limit CPU usage
2019-03-07 16:03:02,127 - INFO - Checking the limits
2019-03-07 16:03:02,127 - INFO - Current CPU limit: 50%
2019-03-07 16:03:02,127 - INFO - Possible new limit would be: 27%
2019-03-07 16:03:02,127 - INFO - Minimum CPU requirement: 10%
```

NOTE:
Docker uses CFS to control the containers CPU usage. The default build of the Raspbian kernel does not contain CFS, therefore the kernel have to be rebuilt with CFS enables, in order to use all the CPU constraint functionality. This test was performed on the cloud node with x86_64 architecture, and Linux operational system.

6.3 Rollback updated service

Introdution:
One of the main problems in releasing a new software is unreliable and incomplete testing. In this experiment, we demonstrate the ability of the QoS monitoring service to detect faulty behavior of a newly updated service and perform rollback to a proven legacy version using CI/CD infrastructure.

Procedure:
For this experiment, we made two images of the service. One with the `latest` tag, this contains the same service as we used in the memory constraint experiment. The other one - with tag `0.1` - contains the same web server without memory leaking function. The implementation can be found at: `https://github.com/tuw-cpsg/smartproduction/tree/master/docker/images/Experiments/rollback_test`. Since we used the same memory leaking service, the Prometheus alerting rule was the same in this experiment, as in the memory leakage.

QoS monitoring and reconfiguration service configuration:

```
"rollback_test": {
        "requirements": {
            "minimum_memory": 120,
            "maximum_memory": 250,
            "can_updated_alone": "True" },
        "solution_priority": {
            "rollback": 2,
            "notify": 1  }
```

As we deployed the service, it starts to use heavily the memory in the same way, as in the memory constraint experiment. When the service reached the limit, Prometheus sent

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the alert. As can be seen from the logs below, the QoS monitoring and reconfiguration service checked the solution order. Since the rollback was the solution with the highest priority, the service checked the available image versions for the container.

The log from QoS monitoring and reconfiguration service:

```
2019-03-07 13:35:48,917 - INFO - identified order: [u'rollback', u'notify']
2019-03-07 13:35:48,918 - INFO - possible solution: rollback
2019-03-07 13:35:48,918 - INFO - Try rollback
2019-03-07 13:35:48,925 - INFO - Check possible tags:
2019-03-07 13:35:48,925 - INFO - possible tags: ['0.1', 'latest']
2019-03-07 13:35:48,925 - INFO - Rollback to image: 128.130.39.67:5000/rollback_test:0.1
```

Since there was a previous version (`0.1`), the service updated the application with that image version. The docker logs about the process can be seen in the Table 6.1:

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
jy1276ipv1ud	rollback_test.1	128.130.39.67:5000/rollback_test:0.1	worker1	Running	Running 46 seconds ago
3yopzxquhw6y	rollback_test.1	128.130.39.67:5000/rollback_test:latest	worker1	Shutdown	Shutdown 48 seconds ago

Table 6.1: Docker logs - Rollback test

6.4 Migrate the service to another node

Introduction:
During this experiment, we wanted to test, what happens if we want to place the service, which leaks resource into another node. Since swarm does the orchestration automatically, we need to add a constraint, which restricts the deployment into the same node again.

Procedure:
For this experiment, we used the same custom service as in the memory leaking experiment. The Prometheus alert config was the same as in the memory leakage experiment.
QoS monitoring and reconfiguration service configuration:

```
{"services":{
        "resource_tester":{
                "requirements":{
                        "can_updated_alone": "True"},
        "solution_priority":{
                "migrate":2,
                "notify":1,}
        }
}
```

As it can be seen in the Figure 6.3, the service is ííííííííupdated. On that Figure, we put both nodes, that is why every service is listed twice. In the Table 6.2, it can be seen, that the service started in the `worker1`, and the QoS monitoring and reconfiguration service migrated it into `worker2`.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

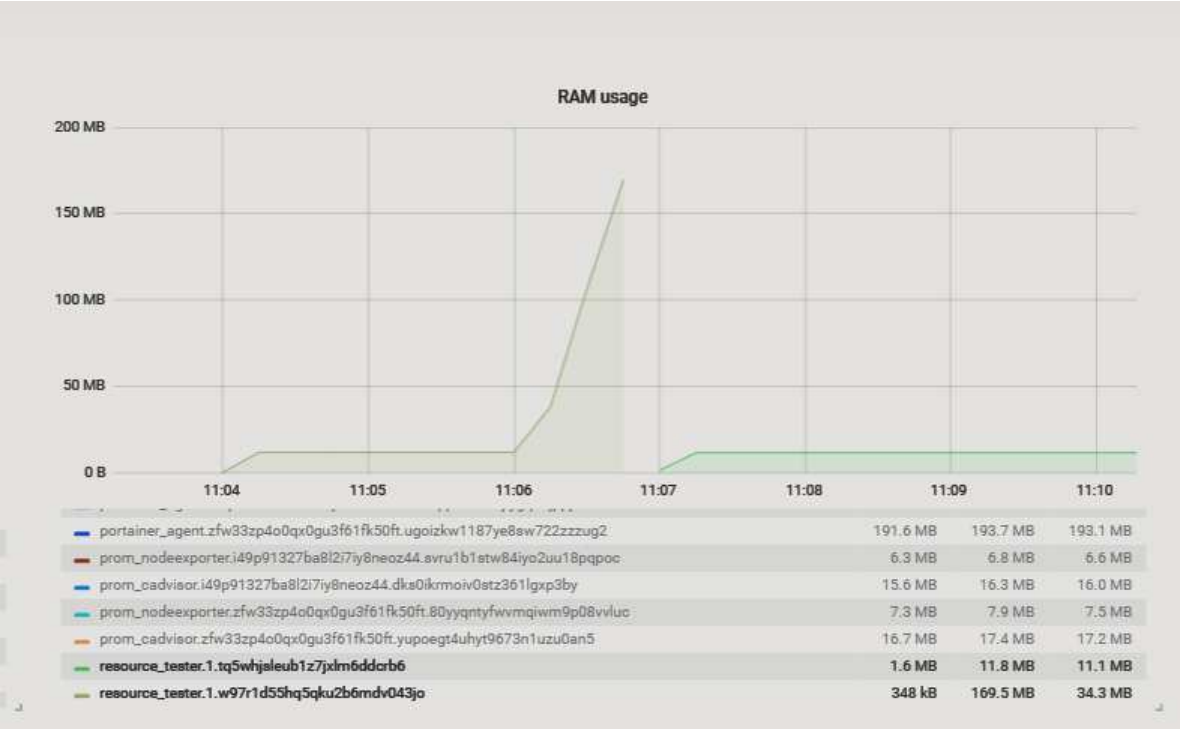


Figure 6.3: Migration of leaking service

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
tq5whjsleub1	resource_tester.1	128.130.39.67:5000/resource_tester:arm	worke2	Running	Running 41 seconds ago
w97r1d55hq5q	resource_tester.1	128.130.39.67:5000/resource_tester:arm	worker1	Shutdown	Shutdown 45 seconds ago

Table 6.2: Docker logs - Migration test

The log from QoS monitoring and reconfiguration service:

```
2019-03-21 11:07:42,127 - INFO - identified order: [u'migration', u'memory_reduction', u'notify']
2019-03-21 11:07:42,127 - INFO - possible solution: migration
2019-03-21 11:07:42,128 - INFO - Try solution migration
2019-03-21 11:07:42,128 - INFO - Check migration
2019-03-21 11:07:42,133 - INFO - Do migration
```

6.5 Leaking service in multiple services environment

Introduction:
This experiment demonstrates, how QoS service detects and handles service failures in a multiple services environment. For this simulation, we used the service with memory leakage error, and we test, if the QoS service can bring the system back to the intended operation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Procedure:
During this experiment, we deployed the core Arrowhead Services and the same custom service as in the memory leakage experiment. The Prometheus alert config was the same as in the memory leakage experiment. The QoS monitoring and reconfiguration service's configuration was the same as in the migration experiment.

The memory usage during the experiment can be seen in the Figure 6.4. At the beginning of the experiment, all the service works normally. Then at time 14:39, the `smart_factory_custom_service` got the POST requests, and it was started leaking the memory. After it had reached the memory limit, Prometheus fired the alert, and the reconfiguration service updated the `smart_factory_custom_service`. As can be seen from the docker logs 6.3, the service was migrated from `worker1`, into `worker2`, as expected.

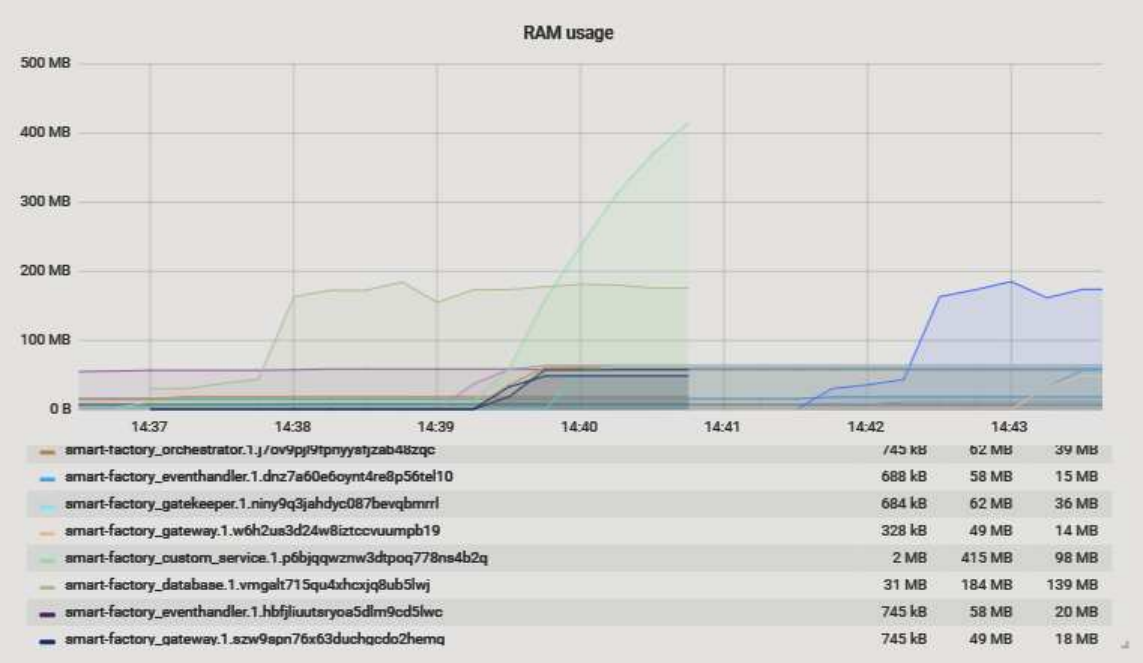


Figure 6.4: Leaking service in multiple services environment

The log from QoS monitoring and reconfiguration service:

```
2019-03-21 14:39:23,24 - INFO - identified order: [u'migration', u'notify']
2019-03-21 14:39:23,24 - INFO - possible solution: migration
2019-03-21 14:39:23,25 - INFO - Try solution migration
2019-03-21 14:39:23,25 - INFO - Check migration
2019-03-21 14:39:23,33 - INFO - Do migration
```

Docker logs can be seen in the Table 6.3:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
l7platdpgjok	smart-factory_database.1	arrowhead-mysql:arm	worker1	Running	Running 5 minutes ago
w6h2us3d24w8	smart-factory_gateway.1	arrowhead-gateway:arm	worker2	Running	Running 5 minutes ago
l4bhm3yicw66	smart-factory_custom_service.1	resource_tester_multiple:multiple	worker2	Running	Running 16 seconds ago
dnz7a60e6oyn	smart-factory_eventhandler.1	arrowhead-eventhandler:arm	worker2	Running	Running 5 minutes ago
p6bjqqwznw3d	smart-factory_custom_service.1	resource_tester_multiple:multiple	worker1	Shutdown	Running 4 minutes ago
niny9q3jahdy	smart-factory_gatekeeper.1	arrowhead-gatekeeper:arm	worker1	Running	Running 5 minutes ago
j7ov9pjl9fpn	smart-factory_orchestrator.1	arrowhead-orchestrator:arm	worker2	Running	Running 5 minutes ago
n26qg0nczx81	smart-factory_authorization.1	arrowhead-authorization:arm	worker2	Running	Running 5 minutes ago
1xk8xlwgs3kg	smart-factory_registry.1	arrowhead-service-registry_sql:arm	worker1	Running	Running 5 minutes ago

Table 6.3: Docker logs - Multiple services test

6.6 Different programming languages

Introduction:
We can implement Arrowhead services many different languages. For this test, we chosen three, namely Python, C++, and Java. In this experiment, we wanted to measure, which one is the best, from performance, and resource utilization point of view. We constructed the experiment based on the assumption that same algorithms with different programing languages produce different performance measurements. We used the Computer Language Benchmarks Game [133] to measure the service performance. They implemented the same algorithm in different programming languages, in order to compare their performance. We choose a regex code since it is a frequent task in the industrial IoT applications. The regex task [134] searches regex patterns in a FASTA format file. FASTA format is a text-based format, for representing different kinds of sequences, such as DNA.

Procedure:
In order to test the performance of these different implementations, we had to create Docker images for each programming language. We used the same base for all images: `balenalib/raspberrypi3-alpine`. We used Python version 3, C++ version 8.2.0 and Java version 8. Since it is a deterministic service (it computes the regex, returns with the result, and stops), we have not defined alerting rules, or reconfiguration configuration, we used the monitoring architecture to constantly record the resource consumption, and check the behaviour of the service. The implementation can be found at [135].

To measure the execution time of the application, we used Linux time command [136]. This returns three measures: "(i) the elapsed real time between invocation and termination, (ii) the user CPU time, and (iii) the system CPU time". To caption the resource usages (CPU, and memory) we used the cAdvisor measurements. The Table 6.4 shows the result.

As it can be seen from the Table 6.4, C++ was the best in this test, while Java and Python performed more similar. However, the test took a little longer for Python, but it used less resource than Java. As we could expected, C++ performed significantly better. Our result is similar with the result from the paper by S. Boragan Aruoba and Jesús Fernández-Villaverde and the Computer Language Benchmarks Game's result.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Language	Memory(MB) low/high/avg	CPU(%) low/high/avg	Time(sec) real/user/sys
C++	0.221/204.9/13.0	0/39.23/1.71	8.812/11.883/0.719
Java	0.221/437/87	0/138/14	43.212/100.404/1.534
Python	0.221/333.9/54.3	0/128/19	74.851/123.993/1.862

Table 6.4: Programming language comparison

(a) CPU (b) Memory

Figure 6.5: Different programming languages - CPP

(a) CPU (b) Memory

Figure 6.6: Different programming languages - Java

(a) CPU (b) Memory

Figure 6.7: Different programming languages - Python

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

As can be seen in the Figures from the monitoring service, the Java application not just used more memory, but for a longer time period. In python case, the high memory usage was a peak - for just a short period of time - similarly to CPP. However, in CPU usage, Python used more CPU for a longer time.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 7

Industrial use-case

In order to test our architecture in a real-world industrial use-case, we implemented an industrial production environment. This environment contains a simulation of a conveyor belt, a data collection and storage service, a database and a real-time dashboard with simulation control ability. In the system, we used the Arrowhead Framework. We used our deployment and monitoring architecture for the building and shipment of Arrowhead services, and the observation of the individual system components. We used the same hardware as was described in the 5.1 section. During this use-case, we can test our architecture with real-world services and workload.

7.1 Simulation

The first component of the industrial use-case is the simulation. This part plays a very important role in our system, due it simulates an industrial equipment, with sensors. In this use-case, we collect the sensor data and control the simulated equipment through a dashboard. To find a perfect solution for the simulation software, we collected some requirements:

- Communication through PLC (Input and Output) via an industrial communication protocol
- Can set variables on the simulated machine and start/stop it through PLC
- Simulated machine has meaningful variables which can be used for later analysis

7.1.1 Industrial simulation platforms

In order to find an optimal software for the industrial use-case, we tried three different simulation platform.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Festo FluidSIM:

Festo FluidSIM [137] is a circuit design and simulation program for pneumatics, hydraulics, and electrical engineering. The company behind this product called Festo is a German multinational industrial control and automation company, they have developed the FluidSim simulation software for over 20 years. Whit FluidSIM many scenarios and machines can be modeled and simulated, which helps to minimize losses due to crashes and ensuring greater efficiency and improved quality. During the implementation of the simulation, you can easily select units from the wide range component library - which are completely according to industrial standards. The selected component that can be drag and dropped to the canvas, and easily connected to each other. The simulator shows the simulated values at run-time. One of the main advantages of this simulation program is that you can build whatever electrical circuit as you want, with any components, and input, output connection. With this tool, there is no case when you can not reach a variable of the simulation, which is a very powerful fact for the later steps, such as the analysis part. However the biggest disadvantage of the simulator is when the user does not comfortable with the electrical circuits, then the building of the first model could be really hard, and the learning curve of the program is really high. To relieve this FluidSim provides many example projects. The other important thing with the simulation is how can you interact with it from another application. This aspect is the key in the CPS/IoT platform, because the simulation will be the data provider, and we have to connect to it somehow. In FluidSIM the connection is implemented with CodeSys.
Controller Development System (CodeSys) [138] is a development platform for implementing PLC applications which are according to industrial standards. CodeSys software is developed by 3S-Smart Software Solutions which is a German software company. This platform helps us to write a controller application for our simulation. CodeSys also provides virtual PLC, which is able to run the controller application. CodeSys platform uses OPC to connect the controller with the simulation.

Festo Ciros

Festo Ciros [139] is a 3D Mechatronics, Robotics and Manufacturing Simulation and Control tool. One of the main difference between these two Festo products - FluidSim and Ciros - is the interface: FluidSim is 2D, while Ciros is 3D. The main focus in Ciros is rather beyond the circuits. Whit this tool you can simulate robots, their movement, etc. In Ciros a whole production line also can be built, to simulate how different robots can interact with each other. Ciros also has example models - such as FluidSim - from the factory and process automation sector to make the platform usage easier. In the simulated robots different programs can be tested, so this is a very cost effective way to learn how to program a factory robot or try different setups for our real-world equipment. The tool also includes an automatic failure feature, which means we can test what happens if some part of the robot has failed. The failure option also a very helpful way to learn how to start a systematically search for a fault in a given machine. As a Festo product it supports the OPC connectivity via Codesys, but here we can have another option namely

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Siemens Simatic Step 7 [100]. Siemens Simatic Step 7 similarly to Codesys, is a software package to support all development phase of a software project. Simatic Step 7 is mainly used with the Siemens S7 controller families.
However, the simulation provides access to many variables, but we can not reach all of them, which makes Ciros less suitable for us.

Siemens Tecnomatix Plant Simulation

Siemens Tecnomatix Plant Simulation [140] is a simulation software with the main purpose of the simulation, visualization, analysis and optimization of production systems and logistics processes. Siemens is a German conglomerate and one of the largest industrial manufacturing company in the world. The plant simulation tool enables the users to analyse, and improve the productivity of their existing production facilities, meanwhile, they can reduce the amount of stock and production time. This analysis is done by simulating the whole production line with assembly stations, and workers. During the simulation, we can see the actual production measurements and the possible bottlenecks. With this feature, it is also possible to validate our new plans and minimize the possible investment risks. Due to this very detailed fashion, it is hard to start the usage of Siemens Plant Simulation, but it also offers some examples, such as the other two. To speak about connectivity: we have a rich collection of possible connections such as SQL, OPC etc. In plant simulation, we also can add random failures to our simulation - such as in Festo Ciros. The main disadvantage of this simulation is that we can not reach the variables of the robots, therefore it is not optimal for our use case.

In the Table 7.1, we summarized the main points of the simulation programs:

Name	Connectivity	User Interface	Robot variables	Automatic failure
Festo FluidSim	OPC with Codesys	2D	Access to every variables	No
Festo Ciros	OPC with Codesys, Step7	3D	Access to many variables	Yes
Siemens Tecnomatix Plant Simulation	OPC with own connector, SQL	3D	Lack of access to robot variables	Yes

Table 7.1: Comparison of the simulators

As it can be seen from the Table 7.1, every simulation have their own disadvantage, but our main criterion was the access to every simulated variables. Since FluidSim won this category, we used this software during the implementation of our system.

7.1.2 Simulation implementation

As it was written, we chose Festo FluidSim, to implement the simulation. In the comparison part, we described that it has a relatively steep learning curve, therefore, for our use-case, we used and modified one of the example projects. In order to collect the data from the simulation, and control it from the dashboard, we used CodeSys virtual PLC. Our simulation architecture can be seen in Figure 7.1.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

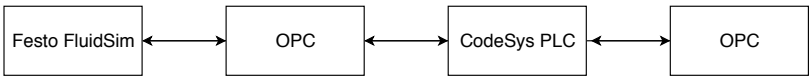


Figure 7.1: Simulation architecture

The chosen example circuit simulates a conveyor belt, controlled by a GRAFCET PLC. In order to control from outside the simulation, we had to change the GRAFCET PLC to CodeSys PLC. For the communication with Codesys, we used FluidSim Input Port and FluidSim Output Port. These ports implement OPC communication within the simulation. In their configuration, we can select connection parameters to the OPC, and OPC items. CodeSys sends and retrieves data through these two modules. The configuration of the Output Port can be seen in Figure 7.2.

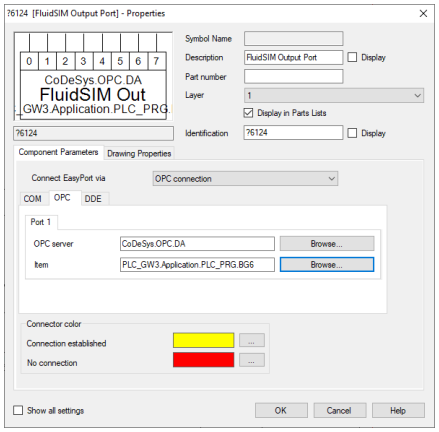


Figure 7.2: Configuration of the FluidSim Output Port

The functionality of this simulation is to transport product on a conveyor belt. The conveyor can be started, stopped, and can be changed the direction of its movement. The control is based on the optical proximity switches on the two sides of the conveyor. This switch closes when the transported product interrupts the light barrier.
The part list of the simulation:

- Conveyor module
- DC permanent-magnet motor
- 2*Electrical connection - 24V
- 2*Electrical connection - 0V
- Pushbutton
- 3*Optical proximity switch
- 3*Relays
- FluidSim Output Port
- FluidSim Input Port

The circuit of the simulation can be seen in the Figure 7.3.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

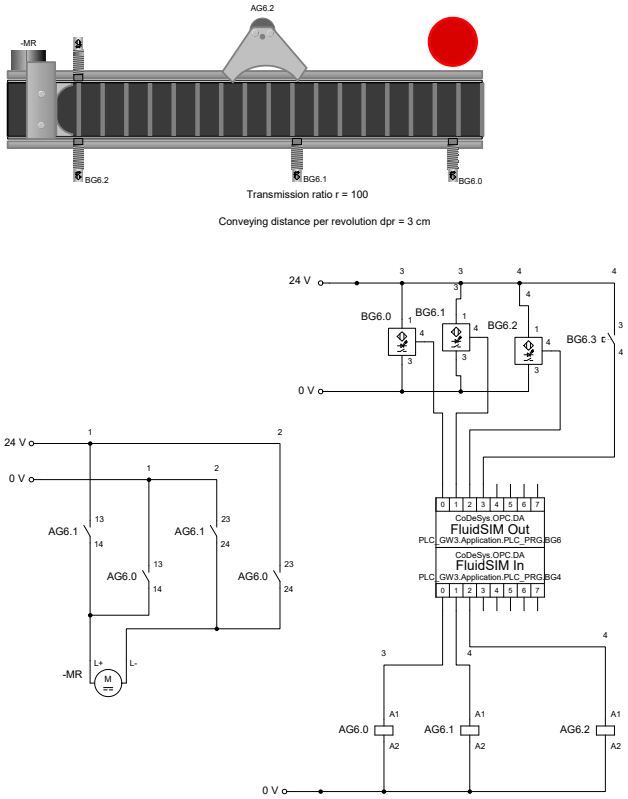


Figure 7.3: Simulation circuit in FluidSim

In order to control the conveyor, we had to implement the control logic in assembly. We used Sequential function chart. The PLC logic algorithm in CodeSys, it can be seen in the 7.4.
The control logic works as follows:
The Init step checks, if there is a product in the conveyor belt. Only one product is allowed at the time. Note, that BG6.2 (the proximity switch in the left-hand side) works inverse, so the value of the light barrier is 0 if something interrupts it, and 1 otherwise. Therefore if there is a product, which interrupts the light at the first switch, and there is no other product in the conveyor(no interruption at the 3. switch), the logic goes to Step0. The BG6 is a BYTE type variable, therefore 13 means that the FluidSimOut module got 1 signal in the number 0, 2, and 3 port. The 0. and 2. means the two proximity switches from the previous step, 3. means the push button, which starts the conveyor. The conveyor moves until the product interrupt the BG6.2, than it changes the direction until it reaches the BG6.1, and so on until it gets stopped. This is a relatively simple logic, but it is perfect for testing the QoS

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

monitoring architecture. The implementation of the simulation can be found at: `https://github.com/tuw-cpsg/smartproduction/tree/master/Simulation`.

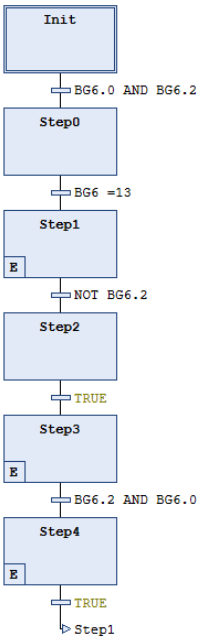


Figure 7.4: PLC control logic in CodeSys

7.2 Data collection

As it was described in the simulation part, we have two main task: sensor data collection and control of the conveyor belt. For these tasks we implemented various Arrowhead services. The first service is the data collection service, with the main purpose of collecting the data from the simulation.

7.2.1 Arrowhead data collection service implementation

In order to extract data from the simulation, we implemented a custom Arrowhead publisher client service (Arrowhead custom service was described in 2.4) in Java. This service has to connect to the OPC server. For the connection with OPC, we used the Eclipse Milo [141]. After we have built a connection with the OPC server, we have to list all the available PLC variable. In order to track the variable values, OPC UA provides a Subscription function [142]. As a client, our Arrowhead producer service can subscribe to the PLC variables, and the server notifies it in case of changes. This mechanism reduces the amount of transferred data, and bandwidth needs. When a new value arrives at the producer service, we make an Arrowhead event from it, and we send it to Arrowhead local cloud. The event contains a timestamp, an event type, and a payload, which is the actual

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

PLC variable. The Arrowhead local cloud than notify all the consumer services, which subscribed to the given type of Arrowhead event. The implementation of the custom producer can be found at: `https://github.com/tuw-cpsg/smartproduction/tree/master/DataCollectionService`.

Algorithm 7.1: Arrowhead data collection service

```
1 InitConnectionToOPCserver;
2 variables := ListAvailablePLCVariables;
3 if variables not null then
4 |   subscribeToVariables
5 |   if variableChanged then
6 |   |   sendToArrowhead(timestamp, eventType, payload)
7 |   end
8 end
```

7.3 Data Storage

As it was written in the 2.1.4 Section, the data generated in an IoT use-case hase some significant attributes. In this chapter we tested and benchmarked three distinct database technology, in order to find the right solution for our system. For this test, we selected a relational database (TimescaleDB), a non-relational database (HBase), and a hybrid database, which is the combination of these two technologies (Kudu).

TimescaleDB:
TimescaleDB [143] is an open-source time-series database. It is engineered from PostgreSQL, therefore it provides full SQL capabilities and interoperability, and reliability. It is compatible with all the client tools as the PostgreSQL. TimescaleDB implemented as an extension for PostgreSQL, with time-series features, and performance optimizations. The main advantage of this, that the same database can be used, therefore it does not require as much development as a new database migration. It supports high data write rate, with in-memory indexes, and batched commits. The data is stored in hyper tables, and partitioned by multiple dimensions - chunks -, such as time interval, or primary key. The biggest disadvantage of TimescaleDB at the moment is that the clustered mode is just in the development phase. Other thing about the TimesaleDB is the community: since it emerged from PostgreSQL, which has a large community, we can find answers fro most of the questions. However, for TimescaleDB specific issues, we can only rely on limited resources, such as the official website.

Hbase:
Hbase [40] is one of the oldest members of the Hadoop ecosystem, as a NoSql database. Hbase - as all the others in this comparison - is open source. It was developed, and modelled after the Google Bigtable. HBase, provide the Bigtable capabilities on top of HDFS. As the other non-relational database HBase also implements horizontal scalability

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- similarly to the Kudu - to support billions of rows in a single table. HBase - as it was mentioned - is an extensible record store, with column-oriented data stores, which is built for low latency, and fast random access. As one of the oldest member, it has a huge community, with many tutorials, forums, and best practices. However one disadvantage of the HBase is the connectivity: it does not support SQL by default, HBase has its own query language. Hbase also provides API-s for Java Python, Scala, C, etc.

Apache Kudu:
Apache Kudu [144] is a part of the Hadoop ecosystem [145], as a storage layer, with the promise of enabling fast analytics on fast data. Kudu is not a typical relational or non-relational database since it has properties from both sides. For instance, it has a columnar storage format which enables fast data analytics with efficient encoding and compression. However, the schema of a table has to be defined before a table can be used. Kudu implements horizontal scalability, via splits a huge table into separate smaller, in different nodes within the cluster. One disadvantage of Kudu is the maturity: it has a relatively small community, with few instructions, and short documentation. Although Kudu provides decent connection possibilities: it has Java, Spark, and Python "NoSql style" API, ODBC, and JDBC connections.

7.3.1 Database benchmark

Introduction:
In order to test these database technologies, we ran a benchmark test on each. For this test, we used Yahoo Cloud Serving Benchmark [146], which is a well-known procedure for evaluating the performance of different storage mechanism. Since we only had one cloud node, and TimescaleDB has no cluster implementation, we tested the databases in standalone mode. The test was performed in the cloud node, with 16G of ram, and Linux operational system on it. We chosen the workload A from the pre-defined core workloads: "this workload has a mix of 50/50 reads and writes. An application example is a session store recording recent actions."

Procedure:
For all database technology we made a docker container. After this, executed the YCSB. The first step of the benchmark, was to load 10.000 rows of data:

```
//Kudu
bin/ycsb load kudu -P workloads/workloada -p kudu\_table\_num\_replicas=1
-p recordcount=10000

//Hbase
bin/ycsb load hbase20 -s -P workloads/workloada -p table=usertable
-p columnfamily=family -p recordcount=10000

//TimescaleDB
bin/ycsb load jdbc -s -P workloads/workloada -cp postgresql-42.2.5.jar
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
-P jdbc-binding/conf/db.properties -p recordcount=1000
```

After the loading, we executed the transaction section:

```
//Kudu
bin/ycsb run kudu -P workloads/workloada -p kudu\_table\_num\_replicas=1
-p recordcount=10000

//Hbase
./bin/ycsb run hbase20 -s -P workloads/workloada -p table=usertable
 -p columnfamily=family -p recordcount=10000

//TimescaleDB
bin/ycsb run jdbc -s -P workloads/workloada -cp postgresql-42.2.5.jar
-P jdbc-binding/conf/db.properties -p recordcount=10000
```

Result:
TimescaleDB performed surprisingly good in this benchmark. One reason for this could be the fact, that we performed the test in standalone mode. This can be a drawback for HBase, and Kudu, since they are designed to run in a distributed environment, and they optimize their queries across the nodes, while TimescaleDB has a more suitable architecture for one node. In the Figure 7.5 the data load can be seen, while Figure 7.6 shows the transaction phase.

After the result, we chose the TimescaleDB, since it seems after the benchmark test, it fits the best in our case. We have only one cloud node, therefore we can not have the benefits with the multi-nodes solutions, such as Kudu, and HBase. We deployed a single node, standalone instance from TimescaleDB with its official docker image. We created the following table structure:

```
TABLE OPC (
time        TIMESTAMPTZ          NOT NULL,
Source      TEXT                 NOT NULL,
BG4         DOUBLE PRECISION     NULL,
BG6         DOUBLE PRECISION     NULL)
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

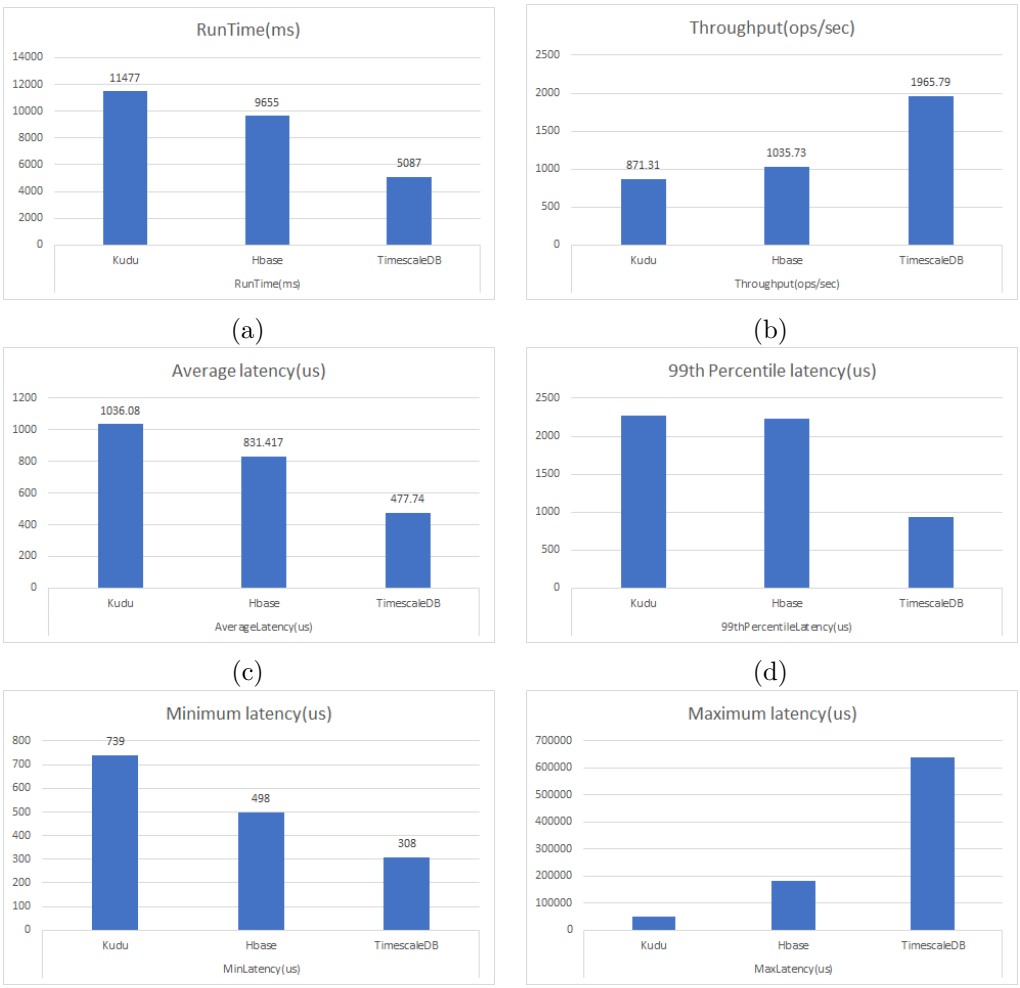


Figure 7.5: YCSB data load

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

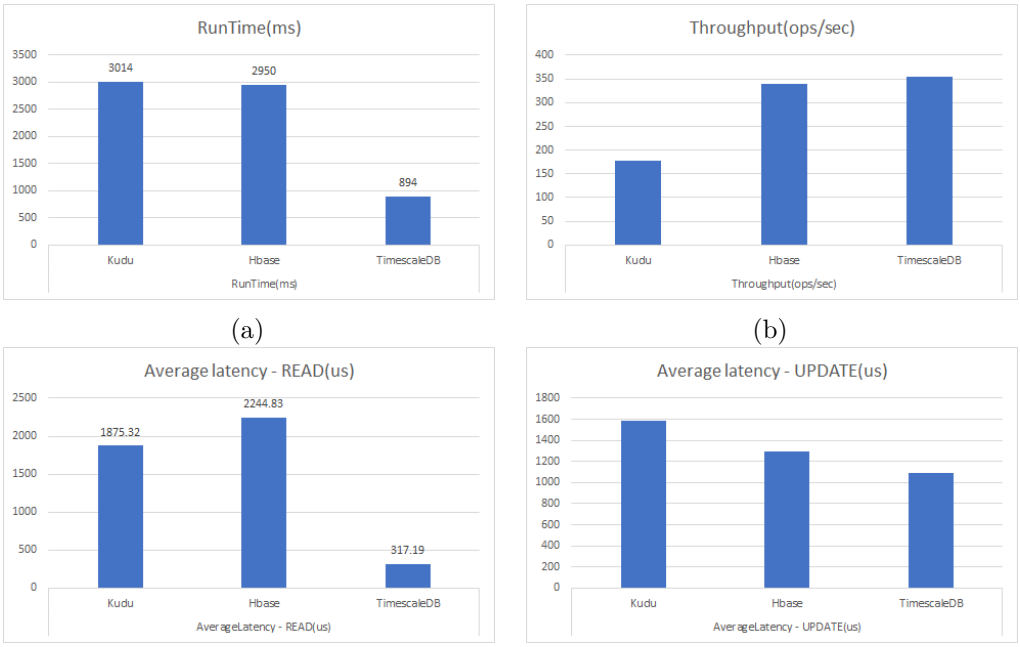


Figure 7.6: YCSB data transactions

7.3.2 Arrowhead data storage service implementation

After selecting the appropriate database, we are able to store the data, collected from the simulation. However, in order to store this data, we had to implement a subscriber type of Arrowhead service. This service is registered to the local Arrowhead cloud. By registering, the service specifies the type of events, it wants to subscribe to. As it was described before, Arrowhead sends a notification to the subscribed services, in case of a matched type of events. The service consumes the event data, parse and stores the values in the database. The implementation of the custom Arrowhead data storage service can be found at `https://github.com/tuw-cpsg/smartproduction/tree/master/DataStorageService`. The implemented algorithm can be seen in the follows:

Algorithm 7.2: Arrowhead data storage service

```
1 InitConnectionWithDatabase;
2 subsribeToSimulationEvents;
3 if message not null then
4 |   value, ts = ExtractValuesFromEvent(e)
5 |   storeValuesInDatabase
6 end
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

7.4 Data visualization, and Simulation control

In order to observe the status and values of the simulation in real-time, we made a user interface for the use-case. As an immediate information source, UI has three functionality: real-time dashboard, data analytics and discovery service, and simulation control service. The UI was implemented in Angular [147]. Angular is a Type-Script based open-source web application framework, developed by Google. The real-time dashboard is based on Grafana, as well as our monitoring solution 2.7.2. Grafana offers refresh periods for the dashboards in regular time intervals, for instance in every 5s. The data source of the charts is the simulation data from the database. In the Figure 7.7 can be seen the dashboard for the conveyor: in the upper left-hand corner you can observe the conveyor actual state whether the conveyor is working or not, with the time from the last failure. In the top chart (BG6), we can explore the state of the three Optical proximity switch, while in the right corner chart (BG4) we can check the direction of the conveyor. If the value of the BG4 variable is 2, the conveyor moves right, if it is 1, than the conveyor moves left.

Figure 7.7: Real time dashboard in CPS/IoT Ecosystem

For the analysis and data discovery purposes, the UI contains a Jupyter notebook [148]. This is an open source development platform, mainly used to implement and compare machine learning applications.
In the control section, we are able to perform instructions in the simulation. In order to do this, we included a form in the UI (in the Figure 7.8), where the user can decide on which sensor value want to change, and the new value for that sensor. Then the backend is generating an Arrowhead message from this form and sends it to the actively running Arrowhead Event Handler.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Figure 7.8: Control panel in the UI

In order to consume the control type Arrowhead events and control the simulation, we implemented another subscriber type Arrowhead service.

7.4.1 Arrowhead OPC control service implementation

This service subscribes to control type Arrowhead messages and establishing a connection with the OPC server. This is the same OPC server, as used in the data collection service. If the UI sends a control message, the Arrowhead framework will notify this service. After the notification, the service has to process the message, parse the OPC Identifier and the value. After the parsing, the service checks if the OPC Identifier is valid or not. If it is valid than the service sets the corresponding value from the message in the OPC server. The change of OPC value than occours changes in the simulations as well, for instance the conveyor shifts direction. The implementation of the custom Arrowhead OPC control service can be found at `https://github.com/tuw-cpsg/smartproduction/tree/master/OPCControlService`. The algorithm of the service can be seen below.

Algorithm 7.3: Arrowhead OPC control service

```
1 InitConnectionWithOPC;
2 subsribeToControlMessages;
3 if new message and message not null then
4 |   parseMessage(OPCId, OPCValue)
5 end
6 if OPCIdIsValid then
7 |   setOPCValue(v)
8 |   changeSimulation()
9 end
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 8

Future work

Although we performed various artificial experiments and implemented a real-world industrial use-case, the future work concerns many possibilities.

The underlying infrastructure allows us to develop monitoring and reconfiguration. In future the goal is to fully integrate STL to Prometheus translation and formal evaluation of models. The integration is important due the transfer from the already existing research result, and system specification in STL. With the integration we can ensure the compatibility and usage of STL with our system.
In this work, for the QoS monitoring and Reconfiguration service we proposed five different corrective actions. We mainly used CPU and memory monitoring for the development and test of the system. However, in the future we can include the other monitored signals, such as network and disk usage in the possible corrective action as well. This service now is a dynamic solution, which means it dynamically decides on the actions to take, however, there is a static way as well. Since every service, and the deployment process has some sort of configuration file, in the static way we make different configurations for different actions. Then we can assign configuration files to the alerts, and redeploy the system based on them.
Prometheus offers a white box monitoring solution, which means, we can observe the internal operation of the services. The other available future improvement is the extent of the Arrowhead Framework services with Prometheus exporter facilities, which makes Prometheus able to pull metrics inside the Arrowhead services. With these capabilities, we can track for instance the incoming Arrowhead messages per consumer instances. This helps us identify possible bottlenecks within the system since we can scale individual service up or down on demand.
Regarding the real-world industrial use-case, we can use more advanced and complicated simulation, or even a real industrial asset to validate our solution. However, initially we concentrated on the industrial area, the suggested solution can serve in the other

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CPS/IoT use-cases as well, therefore there is a potential to test, and validate the system with non-industrial application domain as well.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 9

Conclusion

In this thesis, we proposed a solution for the problem of deployment and monitoring in the CPS/IoT systems. This is a complicated area, because of the special characteristics of its architecture, such as heterogeneous hardware and software infrastructure, the system scale, the minimal avaible computational resource and many more. During the deployment, we have to calculate the best possible endpoint, with the consideration of placement constraints, various architectures, operational systems and service interactions. These difficulties are also presents for the monitoring system, where we have to verify the optimal behaviour of the system in real-time. In order to describe this optimal behaviour, the monitoring solution needs to interpret a system specification in formal languages. For the verification of the constraint violation, the system needs to be able to deal with the real-time data acquisition from all the nodes and services in the system. However, the monitoring system itself cannot restore the optimal state of the system, for this purpose, we need the QoS Monitoring and Reconfiguration service.

During the work, we performed a QoS analysis. This analysis is necessary, since CPS/IoT systems operate in a wide variety of areas with disticnt requirements. All these areas have different QoS requirements, which depend on various factors. For instance, industrial IoT systems have specific security preconditions, while autonomous driving systems are heavily delay-sensitive. QoS factors mostly depends on the use-case, but these factors are not independent. As a result of this analysis, we defined a quality model, which includes all the considerable quality attributes for an CPS/IoT system. With this model, we are able to compare different use-cases based on their QoS requirements.

After the analysis, we implemented a container based deployment and monitoring architecture for CPS/IoT Ecosystem. For a development and experimental environment, we built a hardware architecture, which is organized in three layers, with multiple PCs, and Raspberrys. The software implementation is based on Arrowhead Framework, Docker Swarm, Jenkins, and Prometheus monitoring stack. In order to use Docker Swarm, we had to port all of the used services into Docker containers, which lead us to different

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

problems, for instance with the subscriber type Arrowhead client services. The work contains the result for all the occurred problem.
Through the usage of Jenkins as a continuous integration tool, we had to specify a custom deployment pipeline. This pipeline pulls the latest code of the service from the version control system, then builds a docker image for it. After building this docker image successfully, Jenkins uploads it into a local image registry and deploys it it into a suitable node. In case of any fault during the execution, the deployment pipeline fails, which ensures, that service with any error can not reach the production environment.
As a monitoring solution we used the Prometheus monitoring stack, where we had to configure each service and solve the complication with the different architecture within the cluster. The solution can automatically react to any alternation in the system, such as an additional node, or service, and it starts to pull the metric data without any further configuration. Our custom implemented QoS Monitoring and Reconfiguration service performs dynamic reconfigurations in case of QoS violation within the system. Whether any abnormal system behaviour happen, the monitoring system performs an alert, which will then be received by the Monitoring and Reconfiguration service. This service then analyses the possible actions based on the application domain knowledge. In this work we proposed five possible operation: service's resource reduction (memory, CPU), migrating service to another suitable node, reverting the service to a previous version (rollback), or if any other solution is not possible, we send an alert to a preconfigured developer.
For the validation of our solution, we performed different experiments for each possible scenarios. In order to do this, we created an experiment environment with services, which artificially create QoS violations, such as increasing resource (memory, CPU) consumption. During the experiments, we validated the behaviour of the system, and compared the solution with the pre-defined configuration. At the end of the work, we introduced a real-world industrial use-case, where we can illustrate the usage of the CPS/IoT Ecosystem. This use-case contains a simulation, a data collection and storage service, a control service, a database, and a user interface with simulation controlling ability. During the implementation of the use-case, we performed an analysis for the choice of the used technologies.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

1.1 Basic architecture of a CPS/IoT system 1
1.2 Industrial revolutions timeline. 2
1.3 CPS/IoT architecture heterogen hardwares 4
1.4 CPS/IoT architecture heterogen softwares 5

2.1 Fog architecture 11
2.2 MQTT Architecture 12
2.3 OPC Architecture 14
2.4 Information exchange between Producing and Consuming systems [47] . . 19
2.5 The mandatory, optional Core Systems and other Systems [50] 20
2.6 Arrowhead Framework System of systems [51] 21
2.7 Arrowhead Framework Service Registry [47] 22
2.8 Docker architecture 24
2.9 Swarm architecture 24
2.10 Load balance in Docker swarm 25
2.11 CI/CD Process 27
2.12 Robustness degree 31
2.13 STL monitoring 34
2.14 Prometheus Architecture. Source: `https://prometheus.io/docs/introduction/overview/` 35
2.15 Prometheus data model 36
2.16 Prometheus data model elements 36
2.17 Customized quality model 43

3.1 Architecture for QoS in Arrowhead 53

4.1 QoS Monitoring and Reconfiguration Service 55

5.1 Implementation overview 59
5.2 Hardware architecture 60
5.3 CPS/IoT Ecosystem Docker Swarm Architecture 61
5.4 Wait for it script in usage 63
5.5 Arrowhead subscriber service in Docker swarm 64
5.6 Deployment workflow in CPS/IoT Ecosystem 65

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.7 Monitoring architecture in CPS/IoT Ecosystem 66

6.1 Service start leaking memory 73
6.2 Service with increased CPU usage 74
6.3 Migration of leaking service 77
6.4 Leaking service in multiple services environment 78
6.5 Different programming languages - CPP 80
6.6 Different programming languages - Java 80
6.7 Different programming languages - Python 80

7.1 Simulation architecture 86
7.2 Configuration of the FluidSim Output Port 86
7.3 Simulation circuit in FluidSim 87
7.4 PLC control logic in CodeSys 88
7.5 YCSB data load 92
7.6 YCSB data transactions 93
7.7 Real time dashboard in CPS/IoT Ecosystem 94
7.8 Control panel in the UI 95

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

List of Tables

2.1 Weights for Quality Attributes 50

6.1 Docker logs - Rollback test 76
6.2 Docker logs - Migration test 77
6.3 Docker logs - Multiple services test 79
6.4 Programming language comparison 80

7.1 Comparison of the simulators 85

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Algorithms

2.1 Monitor [72] 32

5.1 Jenkins Docker Image build 65
5.2 QoS monitoring and reconfiguration service 68

6.1 Memory leaking service 72

7.1 Arrowhead data collection service 89
7.2 Arrowhead data storage service 93
7.3 Arrowhead OPC control service 95

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acronyms

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks. 19

ACL Access Control List. 28

CPS Cyber Physical System. 1, 9

CPSN Cyber Physical Sensor Networ. 19

DiffServ Differentiated Services. 20

ETL Extract Transform Load. 73

HMI Human Machine Interface. 66

IntServ Integrated Services. 20

IoT Internet of Things. 1, 7

LAN Local Area Network. 21

MQTT Message Queuing Telemetry Transport. 65

MTBF Mean Time Between Failures. 27

MTTF Mean Time to Failure. 28

MTTR Mean Time Taken to Repair. 28

NFC Near Field Communication. 21

OEE Overall Equipment Effectiveness. 11

OPC Open Platform Communications. 66

OPC AE OPC Alarms and Events. 66

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

OPC DA OPC Data Access. 66

OPC HDA OPC Historical Data Access. 67

OPC UA OPC Unified Architecture. 67

QOS Quality of Service. 2, 19

RDM Relational Data Model. 75

RFID Radio Frequency Identification. 1

RPO Recovery Point Objective. 28

RTO Recovery Time Objective. 28

SCADA Supervisory Control and Data Acquisition. 66

SLA Service Level Agreement. 30

SOA Service Oriented Architecture. 1

wifi Wireless LAN. 21

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Bibliography

[1] A. Čolaković and M. Hadzialic, "Internet of things (iot): A review of enabling technologies, challenges, and open research issues," *Computer Networks*, vol. 144, 07 2018.

[2] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, 2010.

[3] E. Lee, "Cyber-physical systems - are computing foundations adequate?," 01 2006.

[4] L. Ferreira, L. Siksnys, P. Petersen, P. Stluka, C. Chrysoulas, T. le Guilly, M. Albano, A. Skou, C. Teixeira, and T. Pedersen, "Arrowhead compliant virtual market of energy," 09 2014.

[5] M. H. Valipour, B. Amirzafari, K. N. Maleki, and N. Daneshpour, "A brief survey of software architecture concepts and service oriented architecture," in *2009 2nd IEEE International Conference on Computer Science and Information Technology*, pp. 34–38, Aug 2009.

[6] "Arrowhead Framework." `http://www.arrowhead.eu/about/arrowhead-common-technology/arrowhead-framework/`, Accessed: 26-08-2018.

[7] R. K. Naha, S. K. Garg, and A. Chan, "Fog computing architecture: Survey and challenges," *CoRR*, vol. abs/1811.09047, 2018.

[8] L. D. Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on Industrial Informatics*, vol. 10, pp. 2233–2243, Nov 2014.

[9] E. Commission, "Germany: Industrie 4.0." `https://ec.europa.eu/growth/tools-databases/dem/monitor/sites/default/files/DTM_Industrie%204.0.pdf`, accessed 06-11-2018.

[10] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & Information Systems Engineering*, vol. 6, no. 4, pp. 239–242, 2014.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[11] N. Jazdi, "Cyber physical systems in the context of industry 4.0," in *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, pp. 1–4, May 2014.

[12] G. Cheng, L. Liu, X. Qiang, and Y. Liu, "Industry 4.0 development and application of intelligent manufacturing," in *2016 International Conference on Information System and Artificial Intelligence (ISAI)*, pp. 407–410, June 2016.

[13] T. Stock and G. Seliger", ""opportunities of sustainable manufacturing in industry 4.0"," *"Procedia CIRP"*, vol. "40", pp. "536 – 541", "2016". "13th Global Conference on Sustainable Manufacturing – Decoupling Growth from Resource Use".

[14] S. Mubeen, S. A. Asadollah, A. V. Papadopoulos, M. Ashjaei, H. Pei-Breivold, and M. Behnam, "Management of service level agreements for cloud services in iot: A systematic mapping study," *IEEE Access*, vol. 6, pp. 30184–30207, 2018.

[15] B. Littlewood and L. Strigini, "Software reliability and dependability: A roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, (New York, NY, USA), pp. 175–188, ACM, 2000.

[16] "ITU-T Rec. E. 800." `https://www.itu.int/rec/T-REC-E.800-200809-I`, Accessed: 06-02-2019.

[17] K. Rose, S. D. Eldridge, and L. Chapin, "The internet of things : An overview understanding the issues and challenges of a more connected world," 2015.

[18] R. Want, "An introduction to rfid technology," *IEEE Pervasive Computing*, vol. 5, pp. 25–33, jan 2006.

[19] W. Dargie and C. Poellabauer, *Fundamentals of Wireless Sensor Networks: Theory and Practice*. 01 2011.

[20] "Self-driving Cars Will Create 2 Petabytes Of Data, What Are The Big Data Opportunities For The Car Industry?." `https://datafloq.com/read/self-driving-cars-create-2-petabytes-data-annually/172`, Accessed: 09-05-2019.

[21] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, pp. 637–646, Oct 2016.

[22] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, (New York, NY, USA), pp. 13–16, ACM, 2012.

[23] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, *"Fog Computing: A Platform for Internet of Things and Analytics" Flavio Bonomi, Rodolfo Milito, Preethi Natarajan and Jiang Zhu, N. Bessis and C. Dobre (eds.), Big Data and Internet of Things: 169 A Roadmap for Smart Environments, Studies in Computational Intelligence 546*. 03 2014.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[24] P. Varshney and Y. Simmhan, "Demystifying fog computing: Characterizing architectures, applications and abstractions," *CoRR*, vol. abs/1702.06331, 2017.

[25] "Amazon Web Services (AWS) - Cloud Computing Services." `https://aws.amazon.com/`, Accessed: 15-05-2019.

[26] "MQTT." `http://mqtt.org/faq`, Accessed: 08-08-2018.

[27] "ISO/IEC 20922:2016." `https://www.iso.org/standard/69466.html`, Accessed: 08-08-2018.

[28] "OPC Foundation." `https://opcfoundation.org/`, Accessed: 08-08-2018.

[29] "OPC DA." `https://opcfoundation.org/developer-tools/specifications-classic/data-access/`, Accessed: 08-08-2018.

[30] "OPC Classic." `https://opcfoundation.org/about/opc-technologies/opc-classic/`, Accessed: 08-08-2018.

[31] "OPC AE." `https://opcfoundation.org/developer-tools/specifications-classic/alarms-and-events/`, Accessed: 08-08-2018.

[32] "OPC AE." `https://opcfoundation.org/developer-tools/specifications-classic/historical-data-access/`, Accessed: 08-08-2018.

[33] "OPC UA." `https://opcfoundation.org/about/opc-technologies/opc-ua/ua-companion-specifications/`, Accessed: 08-08-2018.

[34] A. Sabtu, N. F. M. Azmi, N. N. A. Sjarif, S. A. Ismail, O. M. Yusop, H. Sarkan, and S. Chuprat, "The challenges of extract, transform and loading (etl) system implementation for near real-time environment," in *2017 International Conference on Research and Innovation in Information Systems (ICRIIS)*, pp. 1–5, July 2017.

[35] H. Fang, "Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem," in *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pp. 820–824, June 2015.

[36] B. Devlin, *Data Warehouse: From Architecture to Implementation*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1996.

[37] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, pp. 377–387, June 1970.

[38] "Redis." `https://redis.io/`, Accessed: 03-05-2019.

[39] "MongoDB." `https://www.mongodb.com/`, Accessed: 03-05-2019.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[40] "Apache HBase." https://hbase.apache.org/, Accessed: 03-05-2019.

[41] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda, "Cyber-physical systems in manufacturing," *CIRP Annals*, vol. 65, no. 2, pp. 621 – 641, 2016.

[42] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. LeeSeshia.org, 2011.

[43] I. Malavolta, H. Muccini, and M. Sharaf, "A preliminary study on architecting cyber-physical systems," in *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ECSAW '15, (New York, NY, USA), pp. 20:1–20:6, ACM, 2015.

[44] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *SME Manufacturing Letters*, vol. 3, 12 2014.

[45] "Systems and software engineering — Architecture description. ISO/IEC/IEEE 42010." http://www.iso-architecture.org/ieee-1471/defining-architecture.html, Accessed: 19-04-2019.

[46] "The Open Group - What is SOA?." http://www.opengroup.org/soa/source-book/soa/p1.htm, Accessed: 19-04-2019.

[47] "Arrowhead Core System architecture." https://forge.soa4d.org/docman/?group_id=58&view=listfile&dirid=185, Accessed: 10-12-2018.

[48] "Arrowhead Wiki." https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Arrowhead, Accessed: 10-12-2018.

[49] J. Delsing, P. Varga, L. Ferreira, M. Albano, P. Puñal Pereira, J. Eliasson, O. Carlsson, and H. Derhamy, *3 The Arrowhead Framework architecture: Arrowhead Framework*, pp. 43–88. 02 2017.

[50] C. Hegedüs, D. Kozma, G. Soos, and P. Varga, "Enhancements of the arrowhead framework to refine inter-cloud service interactions," *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, pp. 5259–5264, 2016.

[51] "Arrowhead Technical architecture." https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Technical_architecture, Accessed: 10-12-2018.

[52] "The Transport Layer Security (TLS) Protocol Version 1.3." https://tools.ietf.org/html/rfc8446, Accessed: 19-04-2019.

[53] "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile." https://tools.ietf.org/html/rfc5280, Accessed: 19-04-2019.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[54] "RSA Cryptography Specifications Version 2.2." https://tools.ietf.org/html/rfc8017, Accessed: 19-04-2019.

[55] "Arrowhead Framework Client Skeletons in Java ." https://github.com/arrowhead-f/client-java, Accessed: 10-12-2018.

[56] "Arrowhead Framework Client Skeletons in C++." https://github.com/arrowhead-f/client-cpp, Accessed: 10-12-2018.

[57] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, 03 2014.

[58] "Linux containers." https://linuxcontainers.org/, Accessed: 15-05-2019.

[59] J. E. Smith and Ravi Nair, "The architecture of virtual machines," *Computer*, vol. 38, pp. 32–38, May 2005.

[60] "Docker." https://www.docker.com/, Accessed: 15-05-2019.

[61] "Docker." https://www.docker.com/resources/what-container, Accessed: 17-03-2019.

[62] "Introduction to x64-assembly." https://software.intel.com/en-us/articles/introduction-to-x64-assembly, Accessed: 17-03-2019.

[63] "ARM architecture reference manual." https://developer.arm.com/docs/ddi0403/e/armv7-m-architecture-reference-manual, Accessed: 17-03-2019.

[64] "Docker swarm." https://docs.docker.com/engine/swarm/, Accessed: 15-05-2019.

[65] N. Naik, "Building a virtual system of systems using docker swarm in multiple clouds," in *2016 IEEE International Symposium on Systems Engineering (ISSE)*, pp. 1–3, Oct 2016.

[66] "Cognitive Complexity." https://www.sonarsource.com/resources/white-papers/cognitive-complexity.html, Accessed: 15-05-2019.

[67] "Jenkins." https://jenkins.io//, Accessed: 20-03-2019.

[68] R. W. Butler and G. B. Finelli, "The infeasibility of quantifying the reliability of life-critical real-time software," *IEEE Transactions on Software Engineering*, vol. 19, pp. 3–12, Jan 1993.

[69] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pp. 46–57, Oct 1977.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[70] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," *FORMATS*, vol. 3253, pp. 152–166, 01 2003.

[71] A. Dokhanchi, B. Hoxha, and G. Fainekos, "Mitl specification debugging for monitoring of cyber-physical systems: Invited presentation at the the first workshop on verification and validation of cyber-physical systems," *Electronic Proceedings in Theoretical Computer Science*, vol. 232, pp. 13–16, 12 2016.

[72] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Nickovic, and S. Sankaranarayanan, *Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications*, pp. 135–175. 02 2018.

[73] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262 – 4291, 2009.

[74] D. Nickovic and O. Maler, "Amt: A property-based monitoring tool for analog systems," in *FORMATS*, 2007.

[75] A. Donzé, T. Ferrère, and O. Maler, "Efficient robust monitoring for stl," vol. 8044, 07 2013.

[76] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," vol. 3253, pp. 152–166, 01 2004.

[77] O. Maler and D. NiăźKović, "Monitoring properties of analog and mixed-signal circuits," *Int. J. Softw. Tools Technol. Transf.*, vol. 15, pp. 247–268, June 2013.

[78] A. Dokhanchi, B. Hoxha, and G. Fainekos, "Metric interval temporal logic specification elicitation and debugging," in *2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pp. 70–79, Sep. 2015.

[79] J. V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia, "Robust online monitoring of signal temporal logic," in *RV*, 2015.

[80] J. V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia, "Robust online monitoring of signal temporal logic," in *Runtime Verification* (E. Bartocci and R. Majumdar, eds.), (Cham), pp. 55–70, Springer International Publishing, 2015.

[81] S. Jakšić, E. Bartocci, R. Grosu, and D. Ničković, "Quantitative monitoring of stl with edit distance," in *Runtime Verification* (Y. Falcone and C. Sánchez, eds.), (Cham), pp. 201–218, Springer International Publishing, 2016.

[82] D. Nickovic and O. Maler, "Amt: A property-based monitoring tool for analog systems," pp. 304–319, 10 2007.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[83] D. Ničković, O. Lebeltel, O. Maler, T. Ferrère, and D. Ulus, "Amt 2.0: Qualitative and quantitative trace analysis with extended signal temporal logic," in *Tools and Algorithms for the Construction and Analysis of Systems* (D. Beyer and M. Huisman, eds.), (Cham), pp. 303–319, Springer International Publishing, 2018.

[84] "Prometheus." https://prometheus.io/, Accessed: 10-04-2019.

[85] "Collect Docker metrics with Prometheus." https://docs.docker.com/config/thirdparty/prometheus/, Accessed: 10-04-2019.

[86] "Prometheus." https://prometheus.io/docs/introduction/comparison/, Accessed: 10-04-2019.

[87] "Cloud Native Computing Foundation." https://www.cncf.io/, Accessed: 10-04-2019.

[88] "My Philosophy on Alerting - Rob Ewaschuk." https://docs.google.com/document/d/199PqyG3UsyXlwieHaqbGiWVa8eMWi8zzAn0YfcApr8Q/edit, Accessed: 10-04-2019.

[89] "Prometheus ALERTING RULES." https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/, Accessed: 10-04-2019.

[90] "Prometheus Node Exporter." https://github.com/prometheus/node_exporter, Accessed: 10-04-2019.

[91] "Google Container Advisor." https://github.com/google/cadvisor, Accessed: 10-04-2019.

[92] "Grafana." https://grafana.com/, Accessed: 10-04-2019.

[93] S. Ali, S. B. Qaisar, H. Saeed, M. F. Khan, M. Naeem, and A. Anpalagan, "Network challenges for cyber physical systems with tiny wireless devices: A case study on reliable pipeline condition monitoring," *Sensors*, vol. 15, no. 4, pp. 7172–7205, 2015.

[94] J. Balasubramanian, S. Tambe, A. Gokhale, B. Dasarathy, S. Gadgil, and D. C. Schmidt, "A model-driven qos provisioning engine for cyber physical systems," 2009.

[95] D. Peng and Y. Ruan, "Ahp-based qos evaluation model in the internet of things," in *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pp. 578–581, Dec 2012.

[96] S. Sivakumar, V. Anuratha, and S. Gunasekaran, "Survey on integration of cloud computing and internet of things using application perspective," 2017.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[97] T. Dillon, V. Potdar, J. Singh, and A. Talevski, "Cyber-physical systems: Providing quality of service (qos) in a heterogeneous systems-of-systems environment," in *5th IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2011)*, pp. 330–335, May 2011.

[98] R. Duan, X. Chen, and T. Xing, "A qos architecture for iot," in *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, pp. 717–720, Oct 2011.

[99] J. Liu and L. Zhang, "Qos modeling for cyber-physical systems using aspect-oriented approach," in *2011 Second International Conference on Networking and Distributed Computing*, pp. 154–158, Sep. 2011.

[100] "Siemens PLC." `https://new.siemens.com/global/en/products/automation/systems/industrial/plc.html`, Accessed: 08-08-2018.

[101] "Omron PLC." `http://www.ia.omron.com/products/category/automation-systems/programmable-controllers/`, Accessed: 08-08-2018.

[102] "ISO 25010." `http://iso25000.com/index.php/en/iso-25000-standards/iso-25010`, Accessed: 17-01-2019.

[103] P. Bellavista and A. Zanni, "Feasibility of fog computing deployment based on docker containerization over raspberrypi," in *Proceedings of the 18th International Conference on Distributed Computing and Networking*, ICDCN '17, (New York, NY, USA), pp. 16:1–16:10, ACM, 2017.

[104] A. Brogi and S. Forti, "Qos-aware deployment of iot applications through the fog," *IEEE Internet of Things Journal*, vol. 4, pp. 1185–1192, Oct 2017.

[105] F. Li, M. Vögler, M. Claeßens, and S. Dustdar, "Towards automated iot application deployment by a cloud-based approach," in *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*, pp. 61–68, Dec 2013.

[106] G. Spanoudakis and K. Mahbub, "Requirements monitoring for service-based systems: towards a framework based on event calculus," in *Proceedings. 19th International Conference on Automated Software Engineering, 2004.*, pp. 379–384, Sep. 2004.

[107] M. Montali, F. Maggi, F. Chesani, P. Mello, and W. M. P. Aalst, "Monitoring business constraints with the event calculus," *ACM Transactions on Intelligent Systems and Technology*, vol. 5, 12 2013.

[108] T. Dillon, V. Potdar, J. Singh, and A. Talevski, "Cyber-physical systems: Providing quality of service (qos) in a heterogeneous systems-of-systems environment," pp. 330 – 335, 07 2011.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[109] L. Miclea and T. Sanislav, "About dependability in cyber-physical systems," in *2011 9th East-West Design Test Symposium (EWDTS)*, pp. 17–21, Sep. 2011.

[110] R. Boncea and I. Bacivarov, "A system architecture for monitoring the reliability of iot," 09 2016.

[111] G. White, V. Nallur, and S. Clarke, "Quality of service approaches in iot: A systematic mapping," *Journal of Systems and Software*, vol. 132, pp. 186 – 203, 2017.

[112] "IEEE 802.11TM WIRELESS LOCAL AREA NETWORKS." `http://www.ieee802.org/11/`, Accessed: 17-01-2019.

[113] A. Malik, J. Qadir, B. Ahmad, K.-L. A. Yau, and U. Ullah, "Qos in ieee 802.11-based wireless networks: A contemporary review," *J. Network and Computer Applications*, vol. 55, pp. 24–46, 2015.

[114] "IEEE 802.11-1997." `https://standards.ieee.org/standard/802_11-1997.html`, Accessed: 17-01-2019.

[115] "IEEE 802.11a-1999." `https://standards.ieee.org/standard/802_11a-1999.html`, Accessed: 17-01-2019.

[116] "IEEE 802.11b-1999." `https://standards.ieee.org/standard/802_11b-1999.html`, Accessed: 17-01-2019.

[117] "IEEE 802.11g-2003." `https://standards.ieee.org/standard/802_11g-2003.html`, Accessed: 17-01-2019.

[118] "IEEE 802.11n-2009." `https://standards.ieee.org/standard/802_11n-2009.html`, Accessed: 17-01-2019.

[119] "IEEE 802.11ac-2013." `https://standards.ieee.org/standard/802_11ac-2013.html`, Accessed: 17-01-2019.

[120] "IEEE 802.11ad-2012." `https://standards.ieee.org/standard/802_11ad-2012.html`, Accessed: 17-01-2019.

[121] A. Le, J. Loo, A. Lasebae, M. Aiash, and Y. Luo, "6lowpan: a study on qos security threats and countermeasures using intrusion detection system approach," *International Journal of Communication Systems*, vol. 25, no. 9, pp. 1189–1212.

[122] "6LowPAN." `https://standards.ieee.org/standard/802_15_4-2015-Cor1-2018.html`, Accessed: 18-04-2019.

[123] M. Albano, P. Miguel Barbosa, J. Silva, R. Duarte, L. Ferreira, and J. Delsing, "Quality of service on the arrowhead framework," pp. 1–8, 05 2017.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[124] "Arrowhead 4.0 System Design Document." https://soa4d.org/docman/view.php/58/218/Arrowhead+G4.0+generic+SoSDD+-+Core+System+architecture.docx, Accessed: 10-12-2018.

[125] "Raspberry Pi." https://www.raspberrypi.org/, Accessed: 17-03-2019.

[126] "Raspberry Pi 3 Model B." https://www.raspberrypi.org/products/raspberry-pi-3-model-b/, Accessed: 08-08-2018.

[127] "Docker compose file for industrial use-case." https://github.com/tuw-cpsg/smartproduction/blob/master/docker/scripts/stack_configuration/smartFactory/docker-compose.yml.

[128] "Control startup and shutdown order in Compose." https://docs.docker.com/compose/startup-order/, Accessed: 15-05-2019.

[129] "Docker swarm monitor implementation." https://github.com/tuw-cpsg/smartproduction/tree/master/DockerSwarmMonitoring.

[130] "Docker swarm monitor." https://github.com/uniba-ktr/docker-swarm-monitor.

[131] "Linux stress manual." https://linux.die.net/man/1/stress, Accessed: 10-04-2019.

[132] "Increased CPU usage experiment." https://github.com/tuw-cpsg/smartproduction/tree/master/docker/images/Experiments/CPU_test/cpu_test.

[133] "The Computer Language Benchmarks Game." https://benchmarksgame-team.pages.debian.net/benchmarksgame/, Accessed: 10-04-2019.

[134] "Regex-redux description." https://benchmarksgame-team.pages.debian.net/benchmarksgame/description/regexredux.html#regexredux., Accessed: 10-04-2019.

[135] "Different programming language experiment." https://github.com/tuw-cpsg/smartproduction/tree/master/docker/images/Experiments/different_programming_language_experiment.

[136] "time(1) - Linux man page." https://linux.die.net/man/1/time, Accessed: 10-04-2019.

[137] "Festo Fluidsim." https://www.festo-didactic.com/int-en/learning-systems/software-e-learning/fluidsim/fluidsim-5.htm, accessed 06-08-2018.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[138] "Codesys." https://www.codesys.com/, Accessed: 08-08-2018.

[139] "Festo Ciros." https://www.festo-didactic.com/int-en/learning-systems/software-e-learning/ciros/ciros-professional-training-in-virtual-learning-environments.htm?fbid=aW50LmVuLjU1Ny4xNy4xOC4xMTEwLjc3OTI, Accessed: 08-08-2018.

[140] "Siemens Tecnomatix Plant Simulation." https://www.plm.automation.siemens.com/global/en/products/tecnomatix/, accessed 06-08-2018.

[141] "Eclipse Milo." https://github.com/eclipse/milo, Accessed: 10-04-2019.

[142] "OPC UA Subscription Concept." https://documentation.unified-automation.com/uasdkhp/1.0.0/html/_l2_ua_subscription.html, Accessed: 10-04-2019.

[143] "Open-source time-series database powered by PostgreSQL." https://www.timescale.com/, Accessed: 03-05-2019.

[144] "Apache Kudu." https://kudu.apache.org/, Accessed: 03-05-2019.

[145] "Apache Hadoop." https://hadoop.apache.org/, Accessed: 03-05-2019.

[146] "Yahoo Cloud Serving Benchmark." https://research.yahoo.com/news/yahoo-cloud-serving-benchmark/, Accessed: 03-05-2019.

[147] "Angular framework." https://angular.io/, Accessed: 03-05-2019.

[148] "Project Jupyter." hhttp://jupyter.org/, accessed 16-11-2018.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.