Extended continuous improvement model for Internet of Things system design environments

Cezary Orłowski, Dawid Cygert and Przemysław Nowak

WSB University in Gdańsk, Gdańsk, Poland

ABSTRACT
The aim of this article is extended for [Orłowski, C., Cygert, D., & Nowak, P. (2020). Continuous improvement process model for supporting the construction of Internet of Things system design environments. In Communications in computer and information science (pp. 424–435). Springer. https://doi.org/10.1007/978-981-15-3380-8_37] proposal build a model of continuous improvement systems design support, the Internet of Things (IoT). The model was based on the authors’ experience gained during the design process IoT nodes. Therefore, the article presents both the research experiment and the extent to which this experiment pointed to the need to design a production environment. This environment is still being developed. To achieve this goal, the article was divided into four main parts. At the beginning a basic project was presented, which goal was to create an IoT node. During its implementation, there were requirements for a system that would be used to monitor the status of IoT nodes and data flow. Therefore, the second chapter presents the functions of the production environment necessary for both designing the nodes and monitoring their condition. While in the paper [Orłowski, C., Cygert, D., & Nowak, P. (2020). Continuous improvement process model for supporting the construction of Internet of Things system design environments. In Communications in computer and information science (pp. 424–435). Springer. https://doi.org/10.1007/978-981-15-3380-8_37], the basic functions were introduced in the presented article the scope was extended by the functionalities currently tested. They were then called the functions of the application enabling platform (AEP) used in IoT to assess their usefulness in improving the monitoring and maintenance processes of IoT nodes.

1. Introduction (purpose and method of implementation)

Creating IoT systems is a complex design process resulting from the need for constant changes arising from user requirements. Such a process limits the possibilities of available development environments. Therefore, the article focuses on the proposal to apply a continuous improvement approach to continuously monitor the state of the design...
environment and adapt it to the changing requirements of IoT systems. This subject was discussed in detail in (Burns & Lachowski, 2018). This article, due to the possibility of full presentation of functionalities used by the authors, allows authors to present a full presentation of the approach used for CI/CD processes.

Continuous improvement is a programming practice which aims to ensure the effective and efficient implementation of the full software development cycle (Burns & Lachowski, 2018; Kurose & Ross, 2018). This means that during the production process the level of the design environment is adapted to the functionality of the IoT system expected by the customer. Achieving this state of affairs is possible by ensuring continuous integration of design environments with the implemented IoT system. This condition is possible thanks to the use of an agile approach, the use of an incremental approach in the production process, ensuring the work of programmers in closed workspaces, as well as the use of a common repository in the design process. Then, after an attack of continuous integration, they create conditions to control product development processes. To create such conditions, it is necessary to create a continuous integration environment and to use feedback mechanisms that allow continuous construction of images with all incremental changes in implementation, as well as their publication when certain criteria are met (Microservices.io, 2019b; Newman, 2015).

The continuous integration process requires building an integration server. An example of such an integration process may be the use of the Analysis and Resolution process according to the maturity model of the CMMI-DEV v1.3 and Jenkins processes (Blog i-systems.pl, 2019; Żeliński, 2019); Jenkins is treated as a tool for implementing Continuous improvement processes due to the possibility of including a wide range of available functionalities. Useful solutions in this case are, for example, version control, management and feedback, analytical and testing poles. Another mechanism necessary for the implementation of the continuous integration process is the feedback mechanism. It provides the ability to automate work in a situation, the emergence of problems with continuous integration. It is important to specify in this case the conditions in which the feedback process takes place, i.e. in such conditions in which changes in the functionality of the production environment occur.

Another of the processes important from the point of view of Continuous improvement is the compilation process, under which scripts are prepared for automating the compilation process. This means that the build processes are started automatically. This allows the team to collaborate easily when creating code and ensures that developers do not forget to install the necessary files. Another process is static analysis, which involves debugging a computer program. It is done by checking the program start code. Static analysis is designed to improve the quality of the code. The next stage is testing processes that are crucial for continuous integration. It is the results of these processes that indicate how much the next software version or the next added functionality can be implemented with the system. It is also a very important process to make sure that new functionalities of the production environment affect the improvement of the produced software. Then unit tests are used as well as automatic tests based on embedded scripts can be used (Ncoder.pl, 2019).

The environmental management process improves continuous integration processes. An example would be using the Jira Project Management environment. It is an integrated system with which you can integrate tasks to Jenkins and direct information about them
to the project team indicating the status of ongoing work. An example of continuous integration processes is shown in Figure 1.

2. Research experiment

The research experiment was aimed at designing and implementing a system that would allow access to data on air pollution to users. It assumed the construction of a portable device that is capable of sending GPS data as well as receiving and displaying data on the built-in monitor. In addition, creation of a mobile and internet application that fulfils the function of displaying received data. It was also necessary to create a system capable of processing the data sent by the device and placing them in the database (Figure 2).

The project group consisted of a dozen or so people – students carrying out the project, mainly outside the university, presenting only the results of the progress of work. The working time was to last about three months. The group has been divided into four teams dealing with: device, mobile and internet application, with backend application and database (Bulldogjob.pl, 2019a; Docker Documentation, 2019). Each team made autonomous decisions regarding the solutions used. Efforts were made to design according to Occam’s razor principle, ‘The simplest solutions are usually the best.’

In the construction of environmental architecture, the following solutions were decided:

- MQTT – data transmission protocol, popular and easy to use, M2M solution. It has a smaller overhead than the HTTP protocol. Due to the popularity of Mosquito, searching for information on, e.g. configuration of secure connections, was not difficult.
PostgreSQL – is a relational database. The project assumed that the data processed by the system would largely be location data. It was decided to use the Postgres database with the PostGIS extension, which allows you to save geographical data directly to the database and use geospatial functions (Bulldogjob.pl, 2019b; Cloudflare, 2019).

Hasura – business layer application. Thanks to it, it was possible to connect to the database, generate a GraphQL schema and set data access permissions via a graphical interface. This solution allows for an integration with an existing database (postgres) and control of access to operations. When connected to the database, it enables the automatic generation of an API in GraphQL based on existing tables and relationships. An additional functionality offered by Hasura are subscriptions that allow notifying listening clients about changed data in real time. In case, it is necessary to perform some procedure outside the database with some changes in the data – it is possible to use webhooks. Webhooks allow to send the data that has changed, with fixed headers to a certain address, which will get further treatment. In the event that the project would involve expanding the system in a different direction, this solution also acts as a data integrator for many sources (something like an API Gateway) with the use of Remote Schemas. Such an approach makes the integration of many GraphQL schemas easy to set up and allows for a modular schema stitching approach.

The Arduino platform was used, using the Arduino UNO base with atmega328p clocked 16 MHz crystal. The device was compatible with the Sim808 overlay. Thanks to this, it was possible to communicate with satellites and send GPS data through the operator’s network to the MQTT server.

Applied fattening data for the quarter. Due to the possible increase in the number of users in the system, it was decided not to simplify the solutions used in connection with the small scale of the project, and to generate the map with air pollution data for districts in our city, instead of responding to each user separately – which would significantly increase the load server. The user, sending data to the MQTT server,
had to previously specify through a micro-service dealing with the location – in which district it is. Now, sending his data to the appropriate channel, he informed about his position and identifier, and wanting to download data on air pollution in the area, he chose the appropriate channel based on the district he had previously received.

- Data insertion. The MQTT server was able to handle a significant throughput of messages, but these also had to be sent to the database. We’ve set up a service that collected inserts, preparing them for a dispatch. After receiving a specific amount or meeting other criteria, we relayed a batch insert further, which translated into a slighter load of the SQL server.

During the implementation of the project, among other issues, the following problems have been discovered:

- Non-scalable MQTT broker – Mosquitto. In the later stages of the project, it turned out that the weakest element in the system was the service dealing with message ingestion – which could be easily improved with a replacement of the MQTT server, which did not allow for clustering and group subscriptions. The proposed solution was sufficient due to the very nature of the experiment, but it would be reasonable to use another broker, designed with scalability in mind e.g. VerneMQ.
- Problems with embedded libraries integration for a prototype device for the Arduino platform. The library used by the team was very extensive. It contained many supported devices on the list, which included the Sim808 module. Unfortunately, after research, it turned out that many people speaking on the Internet had problems similar to those faced by the team when establishing a secure and stable connection through the HTTP client embedded in the standard library of the Arduino platform. Ultimately, the project used manually entered AT commands and a fully unused library.

3. Possibilities of the existing design environment

The analysis of the research experiment carried out, as well as the resulting conclusions, indicated the need to extend the existing design environment. The article (Orłowski et al., 2020) indicates the basic extensions due to the limitations in the number of pages of the article. Currently, the full version of the production environment is the result of the conducted research experiment. Attention was drawn to problems with the use of the MQTT broker, integration of libraries, and the need to look differently than in a situation in which a small, non-scalable IoT system was designed.

Therefore, below, based on the experiment carried out, the possibilities and needs of such development of the design environment are shown. The authors of the article wanted to remove the problems encountered on the one hand, but on the other hand try to create a design environment that would allow designing any number of IT devices, and thus a system that would create conditions for obtaining data from any number of devices.

This chapter indicates the functionality of design environments showing the types of architectures that can be used for designing information systems. This diverse approach to design can be based on docker and containers, and also enable the integration of services if you choose a microservice-based approach. The information contained in this chapter
creates the conditions for extending the existing design environment to be open to any challenges posed by IoT systems (Docs.microsoft.com, 2019; Docs.traefik.io, 2019).

Before presenting the possibilities of developing the design environment, it is necessary to answer the question about the architecture type of the proposed IoT system. There are three patterns with different assumptions: monolithic, assuming the existence of a single server serving client requests, distributed, in which the server is replaced by cooperating, autonomous micro-services and PaaS (Platform as a Service), where services related to virtualization, server and data warehouse support others with a vendor such as Amazon AWS or Microsoft Azure, leaving the developer control over applications and data.

- High Availability – is a popular solution used in distributed systems. It consists in the fact that, in the event of an unplanned failure of one element in the system, its duties are transferred to another element capable of replacing it. It is often necessary to use virtual IP, in the case of replacing an emergency element, the replacement element should be changed its IP address (Figure 3).

- Load Balancing – is used for dispersing requests between many instances that provide the same functionality, while leveraging the resources. Load Balancing is most often software, implemented on the application layer, DNS servers or even hardware, e.g. a multi-layer switch. Load Balancer can operate on different layers of the OSI (ISO Open Systems Interconnection Reference Model) (Swersky, 2019). Distribute traffic: by exchanging frames in layer 2, with the use of special commodity hardware or software, programmatically on ports in layer 4 and based on established rules for HTTP headers in layer 7. Processes related to responding to requests are transferred to multiple servers, which should have their own dedicated network interfaces. Thanks to this solution, we increase the throughput of the entire system in a controlled manner as shown in example (Figure 4) down below.

Figure 3. An application of a high availability reverse proxy application with an automatic switchover [own source].
Figure 5 shows an example of how High Availability and Load Balancer work. The Load Balancer element is able to distribute, between 3 instances, network traffic, increasing the system capacity. However, Load Balancer is a critical point, if a failure occurs, and the designer did not foresee a spare replica, the whole system may malfunction, so it is also important to use High Availability.

In case of an event, when the service must be available worldwide, it is worth to consider the usage of DNS-based load balancing. This will reduce access time, which is critical for many applications deployed globally.

Figure 6 presents a diagram of the Load Balancer operation at the DNS level. In a situation where from the app’s ‘app.6’ is used by the user and the system has no load balancing solutions implemented, it is redirected to a randomly selected server, US or EU. However, in a situation where a DNS server is placed on each of the physical end servers, the user will be redirected to the one closest to his geographical location. The mechanism of action consists of polling the above mentioned DNS servers thanks to inserted NS and CNAME records, and then selecting the one that will respond the fastest. It was assumed that the fastest responding server is the closest, but this is not true in every case, because things such as delay, packet loss or bandwidth or infrastructure failures affect the response speed.

- Connection pooling – the connection pool makes it possible to reduce the load on the element side which receives irregular short connections. Establishing a permanent pool that is constantly maintained will increase reliability, even with significantly higher network traffic. Such a pool is located between the database and the application.
Figure 5. An example application of high availability for a reverse proxy as well as a load balancer [own source].

Figure 6. An example of geographic-like load balancing at the level of Domain Name System [own source].
that performs queries (Figure 7). This particular case can often be further improved with the help of batching, allowing for more sparsely timed updates, and containing larger batches of forementioned transactions.

- **Process Manager** – The designer must secure his system from the side of service failure. In the event that the program encounters a problem and an unplanned shutdown occurs, without using the appropriate tools, it will not be able to restart the process automatically. The designer can set a service program (daemon), executed without having to interact with the user that will restart it. For the runtime environment Node.js there are e.g. PM2, StrongLoop – process managers, which allow you to easily run subsequent instances of our process on each machine core, which also allows for a huge increase in application performance without the need for manual use of the cluster module. It is also possible to use system and create a Unix service that will restart when an error occurs, and start the code itself, using a different library on many cores, if that has not already happened.

- **Server-side pre-processing** – Sometimes, the project team is not able to predict all the functionalities needed to be implemented in the system. The need to add new elements or the possibility of simplification may occur as the project develops. It is often the case that the collected data can be generalized or processed for a wider audience. A good example for this case is the application that gives users the option of retrieving data on the average temperature at the turn of the month, regarding the city of Warsaw, they will have to download a lot of data related to temperatures, and then process them, resulting in the desired effect (Hasura Blog, 2019; Microservices.io, 2019a). If there are many users interested in the average temperature for Warsaw, it is worth considering creating a service that will have processed data on

![Figure 7](image-url)

*Figure 7.* An overview of a simple benchmark performed on various connection pool sizes [own source].
this type of query and will only make it available to interested parties. This will reduce
the load on the client and server that performs the operations for each user (Gonzalez,
2016).
• Containerization of solutions – a container is a standardized software unit. It stores the
code and all its dependencies. It is created on the basis of configuration files describing
your template, i.e. the image it should use, which ports it must display, and other
configuration parameters. By using containerization, designers get rid of the problems
associated with basic manual installation and configuration of systems, and simplify
processes for other teams that use their components when developing software.
The idea is very similar to virtual machines. The most popular container types are
LXC (Linux Containers) and Docker.

It is worth noting that there is a wide range of images, e.g. for an application written for
the node runtime. An image can weigh up to 670 MB, but you can also use an Alpine
image weighing about 60 MB. Thanks to this, it is possible to eliminate many possible
security holes in the components installed on the system and minimize the potential
attack surface by making the system less extensive. You can also use the docker-slim
tool, which will try to slim the distribution further down. As presented on Figure 8
below, the changes can be quite significant and will vary depending on the distribution
and the code dependencies themselves.

Distributions like Alpine can help a lot with trimming down the size of the container
and promote the usage of ‘musl’ in place of ‘glibc’ that is an implementation of the
‘libc’ library focused on speed and minimal size. Another step one could take is to
create a binary with full static linking instead of a dynamic one and place the executable
binary directly in a ‘scratch’ image – that is basically an empty docker base. The binary file

![Figure 8](https://github.com/docker-slim/docker-slim)

Figure 8. An overview of Docker-slim tool run on sample applications as stated in the docs of the
project [own source + https://github.com/docker-slim/docker-slim].
can also be further reduced in size with usage of a compressing tool like ‘The Ultimate Packer for Executables’ called shortly ‘UPX.’ The right solution greatly depends on the provided application but it should stick down to making the docker deployments small and easy.

- Code Generation – there are several main reasons for using code generation. Productivity is one of those aspects. Well, repeating the same steps with other elements or even other projects will save a lot of time and we will be able to devote to other important aspects.

  Code generation can be performed, among others, from the level of UML (Unified Modelling Language) or class schemes. It significantly accelerates the process of software development, organizes it, which also translates into faster TTM (Time to Market) and thus reduces costs. It is also easier to find inconsistencies in naming or action. The generator that we write can use many parsers, e.g. we can design a REST API generator for C# at the beginning, and then add a parser for C which will allow us to change the target language or framework or the library we want to use. A good example is database schemas with which we can generate DAO (Data Access Object) which allows us to easily map objects on elements in the system, and sometimes even entire generation CRUD (Create, Read, Update, Delete) (Fowler, 2016; Hasura Blog, 2019).

4. Directions of project development

On the basis of observations from the implementation of the project and analysis of possible functionalities from the third chapter, which could be used in the design of the production environment, directions of project development were proposed. On the one hand, they are an example of activities in which the state of the existing manufacturing environment and the possibility of implementing new functionalities should be considered. On the other hand, they show how it is possible to use Continuous Improvement processes in which the functions presented in chapter three could be included.

The summary of chapter four is the new architecture of the solution developed as part of the research experiment. It took into account both the state of functionality of the environment proposed in the third chapter, but also a certain vision of project development under which the various functionalities discussed in the third chapter can be attached. Below are indicated two areas necessary for the development of the system proposed as part of the experiment.

- Data storage – Data sets that are not strictly relational and can be easily presented in the form of documents are in many cases better and more convenient to process and handle. However, there are cases where relational data will be more appropriate, and where standard key-value storage is sufficient. In the conducted experiment, the use of a combination of many such technologies would be the most optimal and would allow the use of OLAP databases, paying special attention to the reporting possibilities in the form of cubes.
• MQTT broker scalability – Replacement of the MQTT broker with Mosquitto for solutions such as: VerneMQ, HiveMQ, RabbitMQ due to easier clustering at the time of further development of the project.

• RDBS records scaling – Due to the small scale of the experiment and the fixed geographical position, no available solutions for database clustering or geographical multi-sharing were used. Nevertheless, due to the limit of data writing speed, when there is only one instance of the master type, data such as user position or pollen concentration should be stored in a no relational database. It will definitely simplify the process of implementing BigData solutions in the future.

• Message queues – most tasks that perform services should be performed asynchronously at a particular point in time. It would be reasonable to use a queue system connected e.g. to Redis to process data and perform operations such as sending emails during registration, or total calculations for a given area during the day. These tasks can be commissioned to be completed at a specific time. When you have the queue and the correct configuration, data about the tasks ordered to be performed will not be lost in case of failure of some elements. In addition, thanks to this, the system will be able to delegate some of the work to be carried out, at moments of excessive movement, later. Performing tasks with a higher and lower priority will allow the use of real asynchronous programming methodologies, which will have greater potential for task handling.

On Figure 9 shows a highly-abstract look at the flow of data and the system itself, which was created as an experiment, with minor amendments, which should be included in the next iteration.

Figure 9. Architecture of the design environment taking into account the conditions of the experiment as well as functionaries of the design environment [own source].
5. Application enablement platform systems

Considering the issues presented in the third chapter, as well as analyzing the presented architecture (Figure 10) after considering the case study and the functionalities proposed in the third chapter, the architecture and description of the Application Enablement Platform (AEP) systems was presented. The presentation of these systems is intended to show what the architecture of the systems used to design and monitor the state of IoT systems should look like. The presentation of these systems here is intended to present the final version of the system used for IoT management. With such architecture, we are able to show how much the design environment we design is adequate to create and design such a system. It also creates conditions for us to have a design environment on the one hand and an existing application on the other. That we also answer the question to what extent it is possible to build Continuous improvement processes supplementing the conditions of the design environment and the requirements used for IoT systems.

The AEP system, thanks to the Software Development Kit creates conditions for managing any IoT services. It should be assumed when building such a system that it is possible to monitor the status of individual devices and treat it as support for initiatives aimed at improving the operation of the IoT system. To build an AEP, comprehensive knowledge is needed to demonstrate their potential for use. Because we do not have such knowledge, hence a solution based on the incremental structure of such a system is proposed. The AEP system integrates three groups of processes: system production, maintenance and management. It also creates conditions to ensure the scalability of designed services based on IoT system architects. This approach is extremely important from the point of view of designing the user interface relevant to the user’s use. Figure 4 shows an example of such a system integrating three layers constituting a set of functionalities discussed in chapter three.

![Figure 10. AEP system and its architecture (Gonzalez, 2016).](image)
Their use in the processes of designing processing processes as well as device design and data visualization depends on the type of system. It should be assumed that before proceeding with the Continuous Improvement process, various types of AEP systems will be evaluated in order to choose the one that will constitute the set of functionalities appropriate to the class of the system being designed. There are currently about 300 such systems available.

6. Continuous improvement process model

The Continuous Improvement process model was developed based on the research experiment carried out as well as on the basis of the analysis of functionalities necessary for the design of IoT systems (Orłowski, 2018, 2019). This model takes into account two areas (Figure 11): the area of design environment and the area of functionality of IoT systems. It is between these areas that there should be cooperation analyzed from the point of view of feedback processes, in which the analysis of functionality indicates the possibility of their design using the design environment. You should consider how the support will look for both the design environment (infrastructure and tool support) and how the support for changes in the functionality of the IoT system will look. Nevertheless, the experience gained regarding both the single node design process and the analysis of the functionality of AEP systems indicate that there is a need to treat the IoT system production process as a process in which Continuous Improvement processes will be essential. It is thanks to these processes that it will be possible to iteratively supplement the functionality of the design environment for the needs of designing the IoT system.

7. Conclusions

This paper presents an extended model of continuous improvement processes to support the development of the Internet of Things system design environments. The work Orłowski et al. (2020) presents the initial concept of extension resulting from the article restrictions and applied in the first step of projecting limited design environments. This article contains the full configuration of the environment taking into account both the conditions of the research project as well as the complete scope of functionality of the
design environment introduced to the design of the Internet of Things systems. The system configured in this way creates a full implementation environment for IoT systems. This extended version can be seen as a group of processes targeted at a wide range of IoT system designers interested in continuous improvement processes. These processes are of particular importance in the design of scalable systems, therefore, the example of IoT system design is also an example of the design of scalable systems.

Due to the fact that current systems are becoming more and more complex, therefore, for this complexity it is necessary to design appropriate environments that will enable the production of these systems. For this reason, the authors of the work proposed a solution in which a simple research experiment consisting in the construction of one IoT node was carried out. Then the appropriate system was designed for this node using a simple design environment. Then they performed an analysis of the functionality of design environments for the production of IoT systems, checking the functionality of these environments.

Existing AEP systems have also been analyzed as the most complex IoT management systems. Based on these analyzes, a process model in continuous integration was developed. This model is the answer to the problem of holistic vision of IoT systems. The article proposes a bottom-up approach, which first designs the node and then builds, which is why the node has an appropriate system. Then further nodes are added and the existing system is checked for suitability. Then, processes of analyzing existing system functionalities in the project are carried out to determine which ones will be appropriate for the increased number of nodes. In the absence of such functionalities, AEP systems are tested as the most complex and most useful for visualizing the complexity of IoT systems. Then, on the one hand, we have a design environment, and on the other – a complex IoT system.

The combination of a design environment with a complex IoT system using a process model in continuous improvement streamlines the process of building such environments suitable for the complexity of the system. The word ‘appropriate’ is deliberately given here, which is why the design environment cannot be too complex or too simplified. Its structure from the point of view of individual nodes is not important, and problems with the use of these environments appear when increasing the number of nodes. For this reason, the developed model can be useful in software development processes and is an example of how to manage these processes so that IoT systems can be designed using appropriate design environments for this complexity.

Our experience to date shows that the creation of a perfectly designed system the first time is unlikely, and selected solutions and technologies should be well thought out before their implementation due to further possible complications and technical support. Premature optimization can be a loss in which we can easily lose ourselves at the expense of other important components of the operating system. The solutions used should be designed and applied in a reasonable manner that allows easy expansion and extension of the weight to avoid bottlenecks limiting the system’s capacity.

**Disclosure statement**

No potential conflict of interest was reported by the author(s).
Notes on contributors

Professor Cezary Orłowski heads the IBM Advanced Research Center (IBM CAS) at the WSB University in Gdańsk. Professor Cezary Orłowski’s area of interest are the processes of building the Internet of Things (IoT) network and digital transformation of organizations. He manages the IoT network construction projects (Smart community development project and the project to develop a civic air quality information network in Gdańsk) and digital transformation projects (Warsaw digital transformation policy project together with Deloitte).

Dawid Cygert is a backend developer and a recent graduate of the WSB University in Gdańsk, Poland. Through years of self-directed learning, he has been specializing in full stack web development. In his spare time, Dawid enjoys diving deep into modern web technologies, distributed systems and talking about Go and TypeScript.

Przemysław Nowak is currently studying for master’s degree in computer science at WSB University in Gdańsk and working as system network administrator (2 years). His fields of interest are boardgames, computer networks and programming in Android, in that particular order.

ORCID

Cezary Orłowski http://orcid.org/0000-0001-7716-5522

References

Blog i-systems.pl. (2019). JSON Web Tokens (JWT) – Blog i-systems.pl. Retrieved June 22, 2019. https://blog.i-systems.pl/json-web-tokens-jwt/

Bulldogjob.pl. (2019a). Kubernetes vs Docker Swarm. Which one to choose? Retrieved July 7, 2019. https://bulldogjob.pl/news/611-kubernetes-vs-docker-swarm-ktorywybrac

Bulldogjob.pl. (2019b). Serverless – What is it and how does it work? Retrieved July 10, 2019. https://bulldogjob.pl/articles/1047-serverless-czyn-jest-i-jak-dziala

Burns, B., & Lachowski, L. (2018). Designing distributed systems. Helion SA.

Cloudflare. (2019). What is a serverless microservice? Serverless microservices explained. Retrieved July 10, 2019. https://www.cloudflare.com/learning/serverless/glossary/serverless-microservice/

Docker Documentation. (2019). Swarm mode overview. Retrieved July 7, 2019. https://docs.docker.com/engine/swarm/

Docs.microsoft.com. (2019). API gateway pattern and direct client communication with the microservice. Retrieved July 10, 2019. https://docs.microsoft.com/pl-pl/dotnet/standard/microservices-architecture/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern

Docs.traefik.io. (2019). Traefik. Retrieved July 10, 2019. https://docs.traefik.io/

Fowler, S. (2016). Production-ready microservices. O’Reilly Media, Inc.

Gonzalez, D. (2016). Developing microservices with node.js. Packt Publishing.

Hasura Blog. (2019). The ultimate guide to writing Dockerfiles for go Webapps. Retrieved July 10, 2019. https://blog.hasura.io/the-ultimate-guide-to-writing-dockerfiles-for-go-web-apps-336efad7012c/

Itwiz. (2019). What is NoSQL and how to use nonrelational databases. Retrieved July 5, 2019. https://itwiz.pl/czym-jest-nosql-jak-wykorzystac-nierelacyjnebazy-danych/

Kurose, J., & Ross, K. (2018). Sieci komputerowe. Ujęcie całościowe (7th ed.). Helion.

Microservices.io. (2019a). Microservices pattern: Decompose by subdomain. Retrieved July 10, 2019. https://microservices.io/patterns/decomposition/decompose-bysubdomain.html

Microservices.io. (2019b). The microservice architecture assessment. Retrieved July 10, 2019. https://microservices.io/platform/microservice-architecture-assessment.html

Ncoder.pl. (2019). Docker – Basic idea (virtualization vs. containerization)| nCoder.pl. Retrieved June 20, 2019. https://ncoder.pl/docker-podstawowa-idea-wirtualizacja-vs-konteneryzacja/

Newman, S. (2015). Building microservices. O’Reilly.
Orłowski, C. (2018). Integration driven simulation environment for designing Internet of Things nodes. *Mechanik. SIMP Redakcja Mechanik – Agenda Wydawnicza SIMP*, 91(12), 1154–1156. https://doi.org/10.17814/mechanik.2018.12.208

Orłowski, C. (2019). The use of group decision-making to improve the monitoring of air quality. In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)* (pp. 127–145). Springer Verlag. https://doi.org/10.1007/978-3-662-60555-4_9

Orłowski, C., Cygert, D., & Nowak, P. (2020). Continuous improvement process model for supporting the construction of Internet of Things system design environments. In *Communications in computer and information science* (pp. 424–435). Springer. https://doi.org/10.1007/978-981-15-3380-8_37

Swersky, D. (2019). The hows, whys and whats of monitoring microservices – The New Stack. *The New Stack*. Retrieved July 10, 2019. https://thenewstack.io/thehows-whys-and-whats-of-monitoring-microservices/

Żeliński, J. (2019). Monolithic system or component solutions. *Computerworld*. Retrieved July 1, 2019. https://www.computerworld.pl/news/System-monolityczny-czy-rozwiazania-komponentowe,407905.html