

OpEx Driven Software Architecture a case study

Sebastien Andreo¹, Ambra Calà¹ and Jan Bosch²

¹Siemens AG Technology, Erlangen, Germany

²Department of Computer Science and Engineering, Chalmers University of Technology, Göteborg, Sweden

Abstract

In the last thirty years, the software industry has changed how systems are architected and how systems are distributed. Software is moving from a monolithic architecture and locally installed application to micro-services architecture and applications accessible through the internet. The accessibility over the internet is provided by the emergence of cloud providers like Amazon AWS, Microsoft Azure, or Google GCP. This transformation also impacts the financial structure of software projects, which is moving from capital expenditure (CapEx) to operational expenditure (OpEx). This paper highlights the implication of architecture decisions on a cloud application's operating cost based on two industrial case studies.

Keywords

Operational Expenditure (OpEx), Software Architecture, Cloud-based Application, Financial Model

1. Introduction

In the last thirty years, the software industry moved over different architecture paradigms, from monolith systems to distributed monolith systems, internet-connected systems, and finally, in the last ten years, to internet native systems. The last transformation was possible with the accession of cloud computing to a defacto state-of-the-art technology. As [1] presented, we observed at the beginning of 2010 that organizations started to migrate their internal applications or products to different cloud providers to benefit from the scalability and availability of the infrastructure. Another important motivation was the promise of the cost-effectiveness of cloud computing.


We observed a transition from infrastructure as a service (IaaS) business to the platform as a service (PaaS) business from the cloud provider side. With PaaS, the cloud providers deliver more building blocks to speed up the development of new functionality and to enable product vendors to operate their products as software as a service (SaaS). This transformation is not insignificant for the financial structure of software development projects. Previously the balance between CapEx and OpEx tilted to capital expenditures. Indeed, all the development and testing costs have to be handled by the software producer. In contrast, almost all operational expenses (servers, installation, and update) had to be carried by the software consumer. Now with SaaS, the software products run in the cloud provider's data centers and are based on more complex billing calculations (e.g., pay per use). Thus, OpEx increases for product vendors, and the choices

ECSA'21: 15th European Conference on Software Architecture, September 13–17, 2021, Växjö, Sweden

✉ sebastien.andreo@siemens.com (S. Andreo); ambra.cala@siemens.com (A. Calà); jan.bosch@chalmers.se (J. Bosch)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

of the building blocks assembled to build the product can have a high impact on the product's financial health.

If an organization or practitioners do not systematically consider the OpEx issue, several risks may realize. As the OpEx is a financial indicator, the software system developed by the organization can create pressure on the business model of the organization itself. Indeed, an increase in OpEx will automatically increase the OpEx-to-Sales ratio. The OpEx-to-Sales ratio is an essential insight for the organization to evaluate the organization's profitability. It represents how much each sale costs for the organization. For example, if an organization sells a functionality for 20\$ per month per user, and the OpEx is around 15\$ per month per user, the OpEx-to-Sales ratio obtained is 0.75. The more this ratio increases, the lower the profitability presented by the organization will be.

In other engineering disciplines, the operational cost aspects are taken into account early in the design phases. For instance, in the design of a power plant, the operating costs are calculated before the construction team builds it, and the design is optimized to minimize these. If CapEx represents the first investment in a project life-cycle, OpEx is a recurring cost. Every year, the owner of the system or the service provider will have to support those costs. However, in the software industry, technologies are the main driver for software system architects, while the financial aspect is usually secondary or even completely ignored. With the evolution of cloud-based software applications, OpEx costs cannot be neglected anymore. It is also essential to notice that OpEx costs can vary significantly depending on product usage. By considering operational costs from the early stage of software architecture design, the software provider's financial risk will be reduced.

This paper aims to illustrate how software architecture and design decisions impact the OpEx and, consequently, have a crucial role in maintaining cloud applications' financial health. We want to raise the awareness of software practitioners on the OpEx challenge based on two industrial case studies.

The paper is organized as follows. The next section briefly presents the background, while section III presents the methodology used to execute this study. Section IV highlights the problem within two case studies. Then section V discusses the implications for practitioners and researchers. Finally, we conclude the paper in section VI.

2. Background

Although the relevance of system life cycle cost has been well known in engineering for many years, many complex systems are still planned, designed, produced, and operated, neglecting their total cost throughout the entire life cycle. Usually, the technical aspects are considered first, with the economic aspects deferred until later [2]. However, the increasing complexity of systems calls for an earlier estimation and analysis of life cycle costs. The systems engineering approach, which typically focuses on complex systems, addresses cost estimation and analysis at the beginning of the product development process to identify and quantify risks and evaluate competing systems' initiatives, proposals, and trade-offs. Models for the estimations of economic attributes of system architecture, such as Total Cost of Ownership (TCO) and Return of Investment (ROI), have been largely investigated in the systems engineering community

[3]. The life cycle cost perspective has proven to be most meaningful during the design phase, where the possibilities of cutting down the costs related to operations and maintenance are large [4]. Indeed, despite the insufficient detailed design information, life cycle cost estimates prepared early at the concept design stage can support the project team in analyzing the cost impact of alternatives and making trade-off decisions throughout the system life cycle. In recent years, OpEx models and tools have been developed in different domains, such as the oil and gas industry [5], [6] or building industry [7].

The parallel between the software industry and the oil and gas industry can be surprising as, for instance, the lifetime of a system is grandly different. Building a refinery is not for a couple of months or years, and on the other hand, considering mobile App development tends to born, evolve, and die rapidly. However, there is also another kind of software that should provide long-term support, like energy management software and factory automation. For those types of software with long period support, the same consideration as OpEx should apply.

Managing the cost of the Cloud is not new. [8] proposes an analysis method and tool for the calculation of Cloud TCO and utilization cost. [9] proposes a System-of-Systems (SoS) approach to model cloud infrastructure and services and analyze them from a techno-economic perspective. Another approach is provided in [10] as the TCO model for Cloud Computing Services addressing the indirect and hidden costs of Cloud Computing.

However, as we move to intelligent and connected systems, software architecture trends will change, including their cost estimation models. Those new trends have been defined in [1], which highlights the change of financial models and budgets from CapEx biased to OpEx biased. The trend towards an economy based on OpEx cost models is also addressed by [11]. The authors propose a methodology of modeling a migration in the Cloud at minimal cost or time and claiming the method's applicability to any similar IT project transitioning from CapEx to OpEx. [12] presents case studies putting in evidence the link between architecture decisions and cost as well as the complexity of choosing a cost-effective architecture with the increase of more complex billing models. [13] proposes an integrated framework to estimate the cost of hosting a SaaS application and can be considered a first contribution to linking the architecture and the OpEx. [14] proposes a cost estimation method that uses DAG-based representation and matrix operations to obtain a rapid but not accurate estimation of costs.

Despite the many studies on cloud cost estimation, the software architecture-related factors are too often neglected. Most of the studies concentrate on the operational IT costs or on the cost of migrating to the Cloud. The cost of the system architecture and the technological decisions are mainly ignored. Ten years after the accession of cloud computing to a defacto state-of-the-art technology, there is a lack of methods and tools to help software practitioners design their applications regarding costs.

3. Research Method

This study covers a place at the border of two domains: the technical domain related to system architecture and the accounting domain associated with the operational cost. In business financial accounting, the notions of Capital expenditure (CapEx) and Operational expenditure (OpEx) refer to categorizing the costs to realize some financial optimization. Our research uses

the terms not at a business entity level (firm, organization) but at a software product level. In that way, we propose an interpretation of CapEx and OpEx considering the cost categorization as described in [15]. As a definition basis for CapEx and OpEx, we use those provided by [16]. CapEx refers to the expenses a business incurs to create benefits in the future. For instance, the development of a new feature (R&D costs) is a CapEx. OpEx refers to expenses a business incurs in its day-to-day operations. For example, the run-time costs of the cloud application fall into the OpEx category.

As the research topic: the impact on OpEx of software architecture and design decisions, is under-researched, we conducted this study using an inductive approach. We performed the observation on a real industry system and designed two case studies. The selection of the two case studies was based on typical software architects' activities which fit into two categories: engineering and re-engineering. The design decisions were not influenced by the researcher but were in total control by the development team.

Data Collection. We collected primary quantitative data as the cloud provider offered billing information. The filtering of the resource costs was guaranteed by the resource tagging mechanism of the cloud platform. Before analysis, the gathered data was prepared. The data set was scanned for missing data and outliers or incorrect tagging. We iterated several times to refine the data set.

Data Analysis. This research focuses on the cost aspect of an architectural decision. Still, while architecting a system, other non-functional requirements have to be considered. The financial results of the different designs were always put into relation to expected non-functional requirements. Each solution was proposed and accepted by the product owner about functional and non-functional requirements.

4. Case study

This section presents two case studies. In both cases, the operational costs will be a criterion for the architecture design decisions. The engineering case is developed in section 4.1 envisaging two deployment alternatives for a data analyst workplace, whereas 4.2 proposes a re-engineering scenario of the storage mechanism.

The system under consideration for this case study provides a data analysis environment to develop algorithms that extract insight from measured data like electrical component aging or electrical component losses and visualization for the energy domain. The data analyst has access to several data sources like plant, factory, or building topology, parametrization of functions, signals stored in a time-series manner.

Moreover, several Python libraries are used to plot, transform, and save the user's algorithm results. The application is a cloud application; therefore, all alternatives will compare cloud architectures. As the application is a real product, functional and non-functional requirements will also be addressed but are out of scope in the paper.

4.1. Case 1: Self-managed service vs. managed service

A typical user of the application is a data analyst who wants to develop an algorithm in a convenient IDE. Therefore, Jupyter [17] has been chosen as an environment well established for

data scientists. It is essential to understand how the users typically interact with the application. We defined a typical usage based on the following assumptions. A user works several hours a day (around 8h). All users are located in the same time zone. The computational power required for the algorithms does not require a specific machine setup.

In this case study, we did not specify the system’s scalability requirement in terms of max user count. We kept the focus on our current load, 5 to 20 parallel users. Two deployment alternatives have been envisaged to implement the desired function. On the one hand, a self-managed Jupyter using Kubernetes [18], and on the other hand AWS Sagemaker, AWS managed Jupyter service [19].

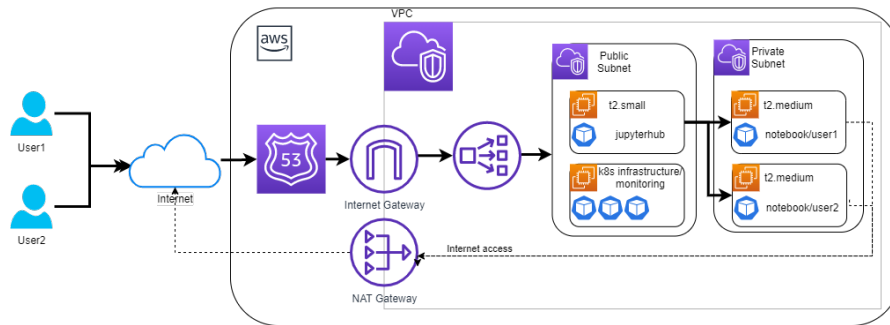


Figure 1: Case 1, Alternative 1 - Architecture overview

4.1.1. Alternative 1.

This alternative proposes a self-managed solution of Jupyter. Self-managed solutions give the owner full control of how the application is managed. However, availability and scalability requirements, for example, have to be implemented. Jupyter can run on a laptop or be deployed on servers, and naturally, there is no restriction for a self-managed solution in the Cloud. [18] describes how to deploy jupyter with the help of Kubernetes, and Figure 1 depicts the IT architecture implemented in AWS by the development team to host and manage the Jupyter environment. The different AWS building blocks required to implement the IT architecture follow several billing models: per million queries/month for Route 53, free for the internet gateway, per GB/month for the NAT Gateway, per hour and per GB transferred for the Elastic Load Balancer and finally per hour for the EC2.

4.1.2. Alternative 2.

Unlike the previous alternative, Alternative 2 proposes a managed solution to deliver the Jupyter environment. Managed services or software as a Service (SaaS) present the advantages and disadvantages as described in [20]. For instance, SaaS tends to reduce installation and maintenance as such activities are part of the service agreement or allows us to focus on our core business instead of technical aspects. However, it also increases the risk of lock-in and reduces customization capabilities. In our case, AWS Sagemaker provided the main component, the Jupyter environment, and the level of customization was sufficient for our needs. Figure 2

depicts the architecture based on AWS Sagemaker Service. The different AWS building blocks required to implement the IT architecture follow both a per-usage billing model. For AWS Sagemaker, it is expressed in per hour/month and per hour/month, and per GB transferred for the NAT Gateway.

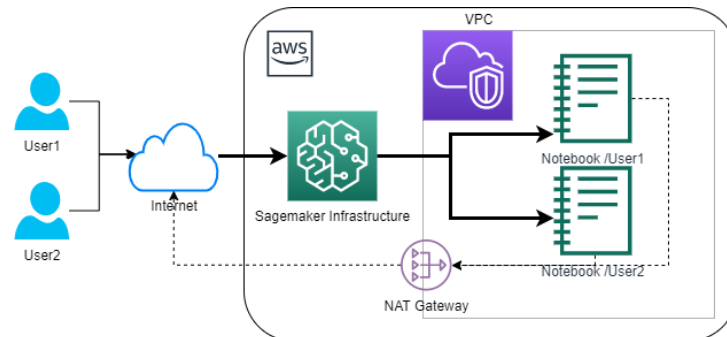


Figure 2: Case 1, Alternative 2 - Architecture overview

4.1.3. Analysis.

Both solutions meet the expected functional criteria. If we make a deep dive into each alternative's cost structure, we can separate the total cost into fixed and variable costs. Indeed some building blocks generate costs, even if nobody uses the application. Such components issue fixed costs in the OpEx. Like the user's EC2 instances, other components are dependent on the application's usage, i.e., variable costs. The cost distribution for each alternative is defined as follows. The first alternative presents a fixed cost of 400\$/month and 7.4\$/user/month, and the second alternative presents a fixed cost of 40\$/month and 8.5\$/user/month. To be able to compare the variable costs of each alternative, we defined a typical user profile as following: a typical user works 8h per day and 20 workdays in a month. The virtual machine types used to execute the application are comparable in terms of cost and capabilities. As a result, we observed a factor ten (10) between both alternatives for the fixed cost and nearly equivalent values for the variable cost. In terms of comparison for the same OpEx, Alternative 1 can deliver the functionality for one user, but Alternative 2 can deliver the same functionality to more than 40 users.

4.2. Case 2: Re-engineering

The application has a connectivity layer that facilitates sensor data collection. The data are collected from different systems, IoT, or cloud systems. Ultimately, its role is to store those data in a structured way to enable a data scientist to extract insight with the development of algorithms. This second case focuses on the re-engineering of the storage system. The product was using the storage architecture described in the next section, and from a functional and non-functional point of view, no issues were discovered. However, as the team on-boards more and more plants, we observed a drastic increase in our storage costs. To meet the product owner

targets of minimum costs, we investigated new technological and architectural alternatives to push the costs down and keeping the functional and non-functional requirements at the same level.

In the next sections, we describe the legacy architecture and then present the re-engineering result to highlight how a technical decision issued an important reduction of the OpEx.

4.2.1. Legacy storage architecture.

Figure 3 shows the architecture of the legacy storage system. The data received from the plant are stored in an S3 bucket within an HDF5 file - a format designed to store and organize large amounts of data. This format is an advantage for our data structure, which may contain thousands of signals. The Panda library used to read the data proposes an HDF5 Reader, but those files cannot be directly read from an S3 Bucket. The EFS component is then used to solve the problem by replicating the data from S3 and providing direct access from the Jupyter notebook.

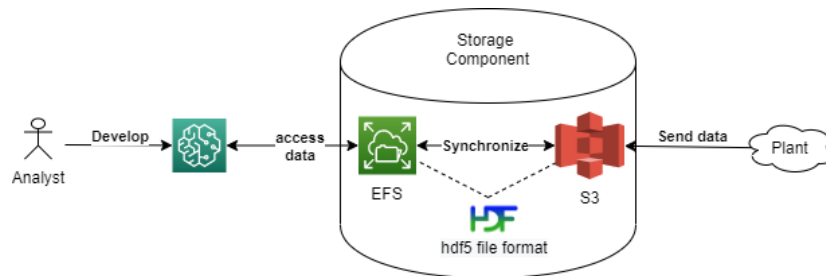


Figure 3: Case 2, Legacy architecture - System context diagram

The AWS components used in this architecture, S3 and EFS, follow a GB/month billing model. We excluded the Sagemaker as it is not part of the re-engineering part. Both components used for storage have a billing model based on the size of the data stored.

4.2.2. New architecture.

Figure 4 presents the revisited architecture of the storage system. The data received from the plant are stored in an S3 Bucket. At that point, the storage file format moved from HDF5 to Parquet. This change's motivation was the ability to read directly from the S3 Bucket using the Panda library and the high compression level of the Parquet file format [21].

The billing model of the unique component S3 is calculated in GB/month. We exclude the Sagemaker as it is not part of the re-engineering part. S3 pricing stays used for storage has a billing model based on the size of the data stored.

4.2.3. Analysis.

Both solutions meet the expected functional criteria. Despite the IO performance difference between S3 and EFS, the users did not perceive tangible differences in accessing data from their Jupyter notebooks. The variable costs, expressed in \$/GB/month, evolved from 0,3845 for the

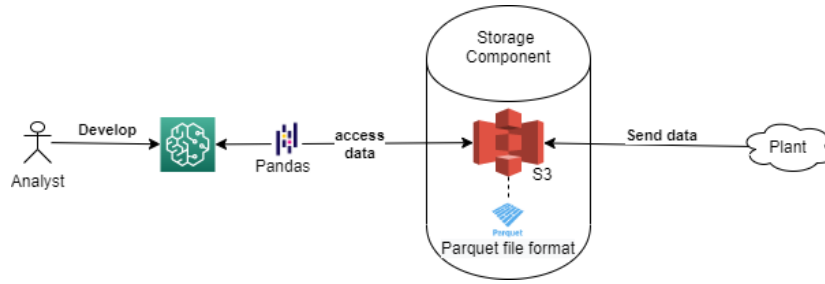


Figure 4: Case 2, Re-engineered architecture - System context diagram.

legacy system to 0,0245 for the Re-engineered. The prices are extracted from the AWS price list for eu-central-1 data center, and we can notice a factor of 15.6 between both solutions in terms of cost.

Table 1

Second case alternatives storage cost evolution between June 2019 and April 2020

	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr
EFS (\$)	482.2	496.6	519.6	559.6	584.2	641.3	697.2	724.2	169.1	8.8	8.8
S3 (\$)	37.6	41.4	27.2	34.7	41	50	67.3	70.6	101.5	92.8	99
Total (\$)	519.8	538	546.8	594.3	625.2	691.3	764.5	794.8	270.6	101.6	107.8

As the first alternative was already implemented, and we performed a re-engineering, we present the storage costs' evolution over 11 months in Table 1. The migration to the second alternative took place between January and March. We can observe a significant reduction of the cost of a factor of 7,8. The remaining EFS costs for March and April are due to the usage of this AWS component for another purpose than storage.

5. Discussion

In this study, we demonstrate that OpEx has a large impact on the software project's financial structure and architectural decisions take a prominent role in controlling the OpEx evolution. Therefore, the cost of architectural or technological decisions should receive much more attention when architecting for the Cloud. The presented cases, although simple, show real industrial applications. The cost impact in both cases was almost an order of magnitude reduction in cost, which cannot be neglected. As the two cases covered different aspects of a system - the first one, an application deployment strategy, and the second one a storage strategy - we are confident that the revealed effect can be generalized to other dimensions of the system like messaging or horizontal scalability. Furthermore, the cost aspect appears as a new quality attribute for the architecture, as for both cases, all architectures were delivering the expected features and the expected performances.

Our experience shows that the presented cases reveal several gaps in the architecture methodology toolbox. The first one is the capacity to map the language or domain model of an application to a cloud provider's language. This mapping is essential to understand and express

the link between the application, for example, storing signals with a specific sampling rate and the cloud resources, dealing with the number of transactions and storage size. This first method is a prerequisite to the next one: cost function definition. In software engineering for a cloud application, the architecture abstracts the code itself and the cloud resources or IT infrastructure. Defining the cost function of a component of an architecture, based on the application language and the cloud resources used to realize this component, would help the architect consider the cost aspect during the design.

There are also some organizational issues to address. For almost twenty years, the agile manifesto advocates the necessity to collaborate more closely with the customers to impact the customer experience positively. As we illustrate in this paper, the software distribution model has changed from selling a product to selling services. We argue along with this paper the importance of considering the OpEx at each phase of the development. Consequently, the software architect should also improve collaboration and exchange with the sales group to understand the selling strategy, the rate plans, and the financial indicator to make the right decision and avoid the decoupling between the business need, the financial constraints and the technology.

To summarize, software architects do not have systematic methods to analyze the cost impact of choice. We think that the interdependency between the business need, technological decision, and cost must be studied in-depth to guarantee the software systems' financial sustainability.

6. Conclusion

In this paper, we highlighted, based on two case studies, the impact of architecture decisions on the OpEx of a software system. This impact should not be neglected, as we demonstrated a difference in cost of around a factor of eight (8) for the two case studies. Other engineering domains have already developed methods and tools to better address operational costs, but there is a lack of awareness about that issue and a lack of methods and tools in the software industry to ensure revenue by design. Since the transformation of applications to cloud-native applications will continue, it is crucial to close the methodology gap and tools gaps. The integration of the necessary modeling activity in the always-moving DevOps and lean culture transition should be taken into account to guarantee a broad adoption of such methodology in the practitioner community.

References

- [1] E. Woods, Software architecture in a changing world, *IEEE Software* 33 (2016) 94–97. doi:10.1109/ms.2016.149.
- [2] Blanchard, B.S., W.J. Fabrycky, *Systems engineering and analysis*, 5th ed. Prentice Hall International Series in Industrial and Systems Engineering (2011).
- [3] M. Nikolaidou, C. Michalakelis, Techno-economic analysis of sysml models, in: *2017 IEEE International Systems Engineering Symposium (ISSE)*, 2017, pp. 1–6.
- [4] E. Sterner, Life-cycle costing and its use in the swedish building sector, in: *Building Research and Information*, volume 28, 2000.

- [5] F. Verre, A. Giubileo, C. Cadegiani, Asset lifecycle opex modelling with montecarlo simulation to reduce uncertainties and to improve field exploitation, in: 4th EAGE North African/Mediterranean Petroleum and Geosciences Conference and Exhibition Tunis 2009, European Association of Geoscientists and Engineers, 2009.
- [6] A. Boccardi, A. Giubileo, C. Cadegiani, New software for opex estimation: Cost driver estimation (code) tool with montecarlo risk analysis simulation, in: SPE Production and Operations Conference and Exhibition, 2010. doi:10.2118/133555-ms.
- [7] D. Geekiyange, T. Ramachandra, N. Thurairajah, A model for early stage estimation of operational expenses (opex) in commercial buildings, in: Proceeding of the 34th Annual ARCOM Conference, 2018, pp. 617–626.
- [8] X. Li, Y. Li, T. Liu, J. Qiu, F. Wang, The method and tool of cost analysis for cloud computing, in: 2009 IEEE International Conference on Cloud Computing, 2009, pp. 93–100. doi:10.1109/CLOUD.2009.84.
- [9] E. Filiopoulou, P. Mitropoulou, A. Tsadimas, C. Michalakelis, M. Nikolaidou, D. Anagnostopoulos, Integrating cost analysis in the cloud: A sos approach, in: 2015 11th International Conference on Innovations in Information Technology (IIT), 2015, pp. 278–283.
- [10] B. Martens, M. Walterbusch, F. Teuteberg, Costing of cloud computing services: A total cost of ownership approach, in: 2012 45th Hawaii International Conference on System Sciences, 2012, pp. 1563–1572. doi:10.1109/HICSS.2012.186.
- [11] T. Antohi, Model for cloud migration cost, 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom) (2019). doi:10.1109/csccloud/edgecom.2019.00014.
- [12] U. Hohenstein, R. Krummenacher, L. Mittermeier, S. Dippl, Towards cost aspects in cloud architectures, in: I. I. Ivanov, M. van Sinderen, F. Leymann, T. Shan (Eds.), Cloud Computing and Services Science, Springer International Publishing, Cham, 2013, pp. 117–134.
- [13] P. Rosati, F. Fowley, C. Pahl, D. Taibi, T. Lynn, Making the cloud work for software producers: Linking architecture, operating cost and revenue, in: CLOSER, 2018.
- [14] T. Aoshima, K. Yoshida, Pre-design stage cost estimation for cloud services, 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC) (2020) 61–66.
- [15] Planview, Inc., What is capex vs opex? - agile costing, 2020. URL: <https://www.planview.com/de/topics/CapEx-vs-OpEx/>.
- [16] Software Advisory, A beginners guide to capex vs opex, 2020. URL: <https://www.softwareadvisoryservice.com/en/whitepapers/a-beginners-guide-to-capex-vs-opex/>.
- [17] J. M. Perkel, Why jupyter is data scientists' computational notebook of choice, Nature 563 (2018) 145–146. doi:10.1038/d41586-018-07196-1.
- [18] L. Resende, On-demand notebooks with jupyterhub, 2018. URL: <https://blog.jupyter.org/on-demand-notebooks-with-jupyterhub-jupyter-enterprise-gateway-and-kubernetes-e8e423695cbf>.
- [19] D. Hudgeon, R. Nichol, Machine learning for business: using amazon sagemaker and jupyter, 2020. URL: <https://aws.amazon.com/sagemaker/>.
- [20] M. Janssen, A. Joha, Challenges for adopting cloud-based software as a service (saas) in the public sector, in: ECIS, 2011.
- [21] I. Zaitsev, The best format to save pandas data, 2019. URL: <https://towardsdatascience.com/the-best-format-to-save-pandas-data-414dca023e0d>.