The Indonesian Journal of Computer Science
www.ijcs.net
Volume 13, Issue 4, August 2024
https://doi.org/10.33022/ijcs.v13i4.4099

Implementation of Cloud Native Architecture in PT. XYZ's Accounting System

Tedy Tri Saputro[1], Rizal Fathoni Aji[2]
tedy@saputro.dev[1], rizal@cs.ui.ac.id[2]
[1,2] Faculty of Computer Science, Universitas Indonesia, Jakarta, Indonesia

Article Information

Received : 7 Jun 2024
Reviewed : 14 Jun 2024
Accepted : 1 Jul 2024

Keywords

cloud-native, microservice, software as a service

Abstract

Microservices are commonly used as an architectural style in SaaS applications. Compared to monolithic architectures, they offer various advantages, such as fault tolerance mechanisms, scalability, and ease of customization. However, microservice architectures cannot stand alone. This is because microservices are based on distributed services and data isolation, and such systems will rely heavily on supporting infrastructure, Therefore, a cloud-native architecture is necessary, taking a philosophical approach to building applications that can fully leverage the cloud-native application model, with the microservice architecture style as one of the key components. This study delves into the integration of cloud-native architecture into the accounting system application of PT XYZ, an IT company specializing in consulting and software integration. The researcher examined the five elements of cloud-native architecture, evaluated them using the twelve-factor app, and identified areas for improvement for PT XYZ.

A. Introduction

Despite the economic pressures stemming from inflation and uncertain macroeconomic conditions in 2023, the projected growth of cloud services in Indonesia presents a promising future. End-user companies are anticipated to spend $679 billion on cloud services in 2024, with expenditures expected to surpass $1 trillion by 2027. This growth not only signifies a significant market potential but also a shift in the technological landscape. By 2028, cloud computing is expected to evolve from a technological disruptor to a critical component for sustaining corporate competitiveness, offering a wealth of opportunities for stakeholders [1].

Indonesia, with its promising landscape, is set to become a hub for cloud computing, offering significant potential for growth. As per Ravenry's analysis [2], Cloud services in Indonesia are on the cusp of a significant growth phase, with a projected increase of $400 million in 2023 and an annual growth rate of 31.9%. This surge is propelled by the increasing number of companies embracing cloud computing, a trend that is expected to drive software spending up by 33% to $900 million in 2023. The landscape is further enriched by the establishment of data centers by world-class public cloud providers like Google Cloud Platform, Amazon Web Services, Microsoft Azure, and Alibaba Cloud, who often collaborate with local companies to promote their services. This promising scenario sets the stage for PT XYZ to seize the business opportunities that lie ahead.

PT XYZ, a local company and an official partner of a leading cloud provider, is making a strategic move. With ten years of experience in IT consulting and software integration, PT XYZ has a product, the Accounting System application, used by several client companies with customization according to client needs. In 2023, the company embarked on a plan to develop accounting system products to be sold to more clients using the Software as a Service (SaaS) scheme. This forward-thinking approach not only aligns with the industry's shift towards cloud-based services but also positions PT XYZ for a more scalable and sustainable future. Under the SaaS model, clients will only pay monthly service fees, while PT XYZ will bear operational costs, maintenance, and infrastructure, a win-win situation for both parties

The current accounting system of PT XYZ, while functional, has its limitations. It consists of two separate applications: a backend and a frontend, both using a monolithic architecture. Each application is deployed on a cloud service using a single instance. If one of the applications fails, such as due to overload, the entire system becomes inaccessible. The system remains down until the operations team manually restarts it. Additionally, the accounting system's scaling process is done manually based on client requests. These limitations underscore the need for a more robust and flexible solution, which PT XYZ is addressing with its transition to the SaaS model.

In response to the product development plans and identified limitations in the existing accounting system, PT XYZ shifted from a monolithic architecture to a microservice architecture. Microservice architecture is a widely used style in SaaS and boasts numerous advantages, such as enhanced fault tolerance mechanisms due to its independent nature [3][4]. It also facilitates easier customization and better scalability than monolithic architectures [5][6][7].

Despite its benefits, developing applications using microservice architecture is more challenging than other architectures. This is because microservice architecture is still a relatively new area of research [8]. Similarly, in the practical world, as illustrated by the Stack Overflow question and answer forum, fewer experts have mastered this field than other topics [3]. For instance, determining the optimal service size is crucial for ensuring the microservice application's Quality of Service (QoS) [8]. Another issue is related to deploying microservice architectures and the mechanisms used to migrate from legacy applications. [9][10][11][12].

One challenge in microservice architecture is designing the cloud infrastructure to support the application. The effectiveness of microservices is closely linked to the underlying infrastructure, and flaws in the infrastructure design can significantly impact the overall architecture. The infrastructure should have self-healing capabilities, fault tolerance, and adaptability to workload changes. Factors like automated testing, DevOps readiness, and continuous integration are also important. Establishing robust infrastructure and automating the application lifecycle is essential due to the complexities of managing distributed systems at scale [13]. This study will explore the cloud architecture used by PT XYZ to support microservice applications in Accounting System products. So, the research question asked in this study is, "What is cloud architecture design for SaaS that meets the business needs of PT XYZ?"

B. Literature Review

Microservices

Microservice architecture is an approach to developing a single application as a set of small services, each running on its process and communicating with each other using lightweight mechanisms. Each service is designed around specific business needs and can be deployed independently using an automated system with minimal centralized management. Additionally, each service can be programmed using different programming languages and utilize various data storage technologies from one another [14]. According to Wu et al. [3], microservices are an architectural style with two key characteristics: decentralization and autonomy. Decentralization means that all services are not centrally managed and controlled. Autonomy implies that each development team can decide on the software or service it develops. This architecture is inspired by SOA's architectural principles, which involve building complex systems as a combination of small, loosely coupled components and communicating between them using neutral APIs [15].

Cloud-Native Architecture

The term "cloud native" was first used in May 2010. Paul Freemantle, a cloud industry expert, wrote on his blog titled "Cloud Native," explaining that cloud-native applications are specifically designed to run on the cloud and have properties that benefit from all the advantages of a cloud environment [16], [17]. Moving a traditional application to a cloud environment does not make the application a cloud-native application [16], [18]. According to Toffetti et al. [18],

cloud-native applications should possess critical characteristics. These include the capability to foresee failures and fluctuations in the quality of cloud resources and third-party services. Moreover, they should be able to quickly adapt to changes in workload and effectively utilize dependable infrastructure.

According to the CNCF definition, Cloud-Native Applications do not depend on a specific technology as long as they comply with the CNCF's three main points [17]:

1. Platform: cloud-native applications can run on various environments, including public, private, or hybrid clouds.
2. Properties: The nature of cloud-native applications is designed to be scalable, loosely coupled, resilient, manageable, and observable.
3. Practices: Practices in cloud-native applications include automation, continuous delivery, and DevOps.

Meanwhile, Reznik et al. [19], state that cloud-native architecture adopts five architectural principles along with two cultural principles.

1. Containerization. Encapsulate the application, its dependencies, and its operational environment in a single package, making testing, moving, and deploying easy.
2. Dynamic management. Using cloud-based servers that offer flexibility
3. Microservices. Design the application as a collection of separate small service components. Each microservice can be deployed, upgraded, scaled, and restarted independently of other services in the application without affecting the end user. Microservices enable increased development speed achieved by parallel component development.
4. Automation. It involves replacing manual work with scripts or code to automate tasks such as maintenance and updates.
5. Orchestration. This process entails integrating all the principles by automating containerized applications' deployment, scaling, and management. For instance, it involves using Kubernetes or other orchestration tools to manage and automate tasks such as container availability, provisioning, and deployment, load balancing of containers across infrastructure, and scaling up or down by adding or removing containers as required.

The two cultural principles are as follows:

1. Delegation. Provide each technician with the necessary tools, training, and autonomy to securely implement changes, deploy, and monitor independently. This means they should be able to make changes without involving other teams or obtaining slow management approvals.
2. Dynamic strategy. Communicate the plan to the team and remain open to adjusting it based on results.

In addition to these components, cloud-native applications should adhere to 12 guidelines and best practices known as the Twelve-Factor App Manifesto. Initially introduced by Heroku engineers, these principles are now widely recognized as a crucial foundation for building cloud-native applications [20] [17]. The principles outlined in the twelve-factor app manifesto are as follows

1. Codebase. There is only one codebase per service, but it can be deployed to different environments.

2. Dependencies. The application's dependencies should be explicitly declared and made available for the dependency manager to download from a central repository. Using containers can reduce issues with dependencies.
3. Configurations. The configuration should be separated from the source code. When the configuration changes, the code should not need to change, and the build process does not need to be done from scratch.
4. Backing services. A backing service refers to an external resource used to support an application's functionality. It is considered an attached resource and can be easily replaced based on the environment without modifying the source code.
5. Build, Release, and Run. The source code should undergo various stages before entering the production environment. It is recommended that automated build processes be used through CI/CD practices.
6. Stateless Processes. To ensure scalability, the application should be run in a stateless process
7. Data Isolation. This is an essential pattern in microservices, where each service should manage its data so that other services cannot access data directly from other services but through APIs from that service.
8. Concurrency. The scaling process can be applied to independent and horizontal services.
9. Disposability. Built services should be easily turned on and off as needed. For example, if there is a failure, the instance should be able to be quickly terminated and launched. Additionally, the services should allow for easy scaling out and down as required.
10. Environment Parity. Keep all company environments as similar as possible. Containers can overcome inconsistencies between environments.
11. Logs. Logging is crucial in distributed systems. Ensuring components provide correct data for remote monitoring is essential to managing them.
12. Administrative Processes. Specific tasks, such as database migration, batch jobs, maintenance jobs, and CI/CD processes, should be executed as one-off processes. Once these tasks are completed, the associated service can be terminated. Additionally, the source code for administrative tasks should be tracked and stored in version control.

Relationship Between Microservices and Cloud Native

Because microservices is a software design pattern, microservice applications can be deployed on traditional infrastructure or in a cloud environment, just like monolithic applications, which can also be deployed to cloud environments or conventional on-premises infrastructure. Cloud-native architecture is a comprehensive approach to building applications that fully utilize the cloud computing model, and microservices are one of the reference implementations of this architecture [19].

The benefits of a microservice architecture can be challenging to realize without proper implementation of cloud-native architecture elements. Designing a

microservice architecture requires careful consideration of infrastructure design to support development and operations. For instance, infrastructure based on microservice architectures should be self-healing, fault-tolerant, and capable of adapting to changes in load. On the development side, adjustments should be made to accommodate automation testing, DevOps readiness, and continuous integration for effective management of microservice applications [13].

Previous Research

Microservice-based application development is often considered more complex than applications with other architectures and has a broad scope. Therefore, substantial research exists on case studies and methodologies for converting monolithic architectures to microservices [11][21][22]. The research emphasizes building microservice applications and lacks discussion of the underlying infrastructure design. Research by Chen T [23] reviews DevOps infrastructure, integrating GitLab, Jenkins, Docker, Kubernetes, and Harbot in a private cloud for automation and maintenance. Meanwhile, Bharadwaj et al. [24] Discusses the cloud-native approach and compares it to the traditional approach for designing, building, and deploying applications. The variety of available technologies has prompted Roselier et al. [25] to develop machine learning and data science-based tools for automating and simplifying the consulting process in selecting cloud-native designs.

Several researchers have conducted studies that did not discuss the practical implementation of cloud-native architecture in industrial environments, particularly in public cloud settings. The author's research addresses this gap by examining PT XYZ's experience in establishing DevOps infrastructure on AWS and GCP to support microservice architecture for accounting system software

C. Research Method

This research is qualitative and aims to explore the architecture of the cloud infrastructure used by PT. XYZ compares it to Cloud Native architectural principles based on CNCF and evaluates them based on Twelve-Factor Apps. In the initial stage of this research, we will prepare background information and identify problems by collecting initial data to describe the company's profile and conditions. Our specific focus is to investigate the architecture of the DevOps infrastructure used by PT XYZ to support the Accounting System application. The subsequent steps will involve conducting a thorough literature review and in-depth research on the principles of cloud-native architecture based on CNCF.

The findings from the literature review will be applied in the result and analysis chapter, where we will analyze the architecture implemented by PT XYZ and compare it with the principles of Cloud Native architecture. In this section, the researcher was assisted by a cloud engineer leader in exploring the infrastructure that PT XYZ uses to support its accounting system. The summary section of this research presents a conclusion of the different findings from the previous chapter, providing recommendations for the subsequent improvement of a cloud infrastructure architecture to enhance the operation of the Accounting System application for PT XYZ.

D. Result and Discussion

In this section, the author interviews technical cloud leaders about the cloud services used by PT XYZ to support its operations. The author analyzes these infrastructure services based on five technical principles of cloud-native architecture: containerization, dynamic management, microservices, automation, and orchestration. Furthermore, the author will assess it according to the Twelve-Factor App principles

Implemented Architecture

1. Containerization

All business services related to accounting systems use Linux containers and are deployed using Cloud Run, while some services associated with CI/CD and Jenkins still use VMs. This decision is based on technical and economic considerations from PT XYZ. Jenkins is a stateful application, and from an operational perspective, it is not yet possible to use Cloud Run. Additionally, using Kubernetes, such as GKE and EKS, has a higher cost than using VMs through Amazon EC2. Table 1 Provides a complete list of services that are identified using containers and VMs

Table 1 List of services using containers and virtual machines.

Services	*Container/VM*	*Image/OS*
Backend Services	Container	jre-alpine
Frontend Services	Container	nginx-alpine
SonarScanner	VM	Ubuntu
Selenium	VM	Ubuntu
JMeter	VM	Ubuntu
Maven	VM	Ubuntu
SonarQube	VM	Amazon Linux

2. Dynamic Management

PT XYZ utilizes two public cloud services: AWS and GCP. AWS is used for the CI/CD infrastructure with Jenkins (Master) and SonarQube deployed on Amazon EC2 services. Meanwhile, GCP is used in microservices for Jenkins Agent and accounting system applications. Each business service from the Accounting System is deployed into Google Cloud Run.

Google Cloud Run was chosen as the computing resource for deploying microservice applications because it offers a container-based serverless environment, which is considered cheaper than using GKE; this is possible because Google Cloud Run bases the billing cost on the number of API requests. Also, the decision to choose Google Cloud Run is based on the fact that Cloud Run provides a cloud-neutral environment. This means that companies are not locked into one vendor (vendor lock-in) and still have the freedom to switch cloud providers if needed.

Details of the cloud services used by PT XYZ, as presented in the Table 2, show that all of the services used by PT XYZ. Some services used are container as a service (CaaS), such as Google Cloud Run, a serverless service. In a serverless architecture, users don't need to manage the underlying infrastructure of Google

Cloud Run. Some technologies related to CI/CD and using Jenkins are deployed on Amazon EC2 services, which is a type of infrastructure as a Service (IaaS). In contrast, some other services that utilize the Jenkins agent, such as Maven, Selenium, and Jmeter, are deployed to Google Cloud Engine. The use of Google Cloud Engine rather than Amazon EC2 for services used by Jenkins agents is based on practical considerations to facilitate the deployment of all business services used for the accounting system, which is deployed to Google Cloud Run

Table 2 Cloud services used

Category	Services	Cloud Service	Operational
CI/CD	Jenkins	Amazon EC2	24 hours
Static code analysis	SonarQube	Amazon EC2	24 hours
Build tools	Maven	Google Compute Engine	On-demand
Automation functional test	Selenium	Google Compute Engine	On-demand
Performance test	Jmeter	Google Compute Engine	On-demand
Container build	Google Cloud Build	Google Cloud Build	On-demand
Container registry	Google Container Registry	Google Container Registry	On-demand
Backend and Frontend service	Spring Boot and VueJS	Google Cloud Run	On-demand
Database	PostgreSQL	Google Cloud SQL	Managed-service
Monitoring	Google Cloud Monitoring	Google Cloud Monitoring	Managed-service
Configuration	Spring Cloud Config	Google Cloud Run	On-demand
Gateway	Spring Cloud Gateway	Google Cloud Run	On-demand

Some services, such as Jenkins and SonarQube, which use Amazon EC2, run continuously for 24 hours. On the other hand, some services that use Jenkins agents launch upon receiving a request and then terminate after a certain period of idleness. This configuration can be achieved by setting the node retention time within the Google Compute Engine plugin installed on Jenkins. In services using Cloud Run, the business service will only run when it receives a request from the client by default

3. Microservice

As illustrated on Figure 1, the Accounting System utilizes a microservice architecture constructed with the Spring Cloud framework for the backend and Vue JS for the front end. The microservice application consists of 3 supporting services and three business services. The supporting services include gateways, authentication, and configuration that facilitate the operation of microservice systems. The main application comprises three business services: inventory, accounting, and purchasing. The number of business services being developed will continue to increase as the application development process continues. Some other services are managed by GCP, including Cloud SQL (a database as a service) and Cloud Monitoring (which monitors and checks the service condition).

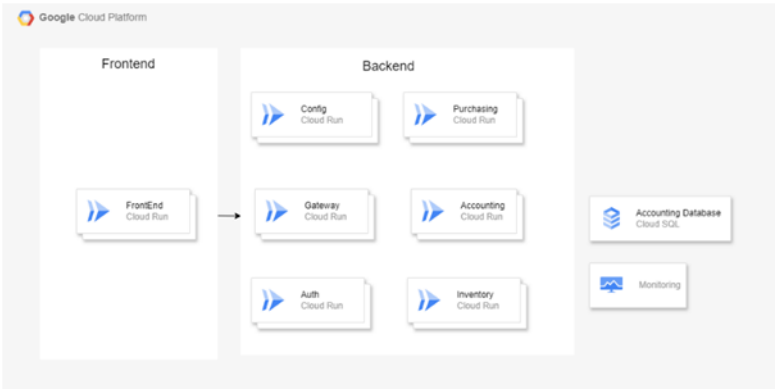

Figure 1. PT XYZ Microservice Application

Every request from the frontend application to the backend must first go through the gateway. At the gateway, the service undergoes the authorization and authentication process with the auth, and then it is forwarded to the business service, which includes purchasing, accounting, and inventory functions. The Configuration service is utilized to externalize configurations. Each time the service runs, it will read the configuration from the Config service.

4. Automation

The diagram of the automation includes the CI/CD infrastructure shown in Figure 2. Jenkins controls the CI/CD process, divided into three stages. The first stage starts with the developer who wants to integrate their source code changes into the development branch; this process is called a pull request. The pull request process triggers Jenkins to perform a static code analysis using SonarQube to check the code quality and unit tests. The technical leader and developer will be informed about the results of the process. The technical leader will manually review the submitted source code to ensure it complies with the functional requirement. After the technical leader merges the developer's requests, they will proceed to the second stage. This involves running the CI/CD pipeline to execute the build process and perform unit tests. Following this, the containerization process and deployment to the Cloud Run environment will occur

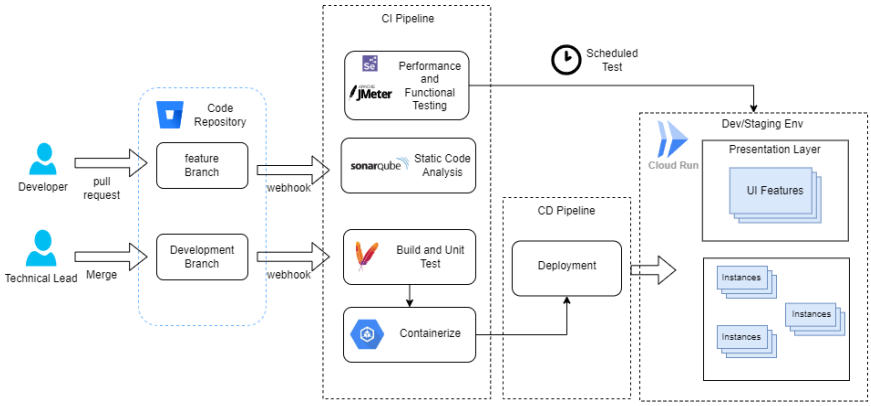

Figure 1 CI/CD infrastructure

The third stage, which involves functional and performance testing, is scheduled in the development/staging environment using Selenium and JMeter. The QA engineer develops the scripts for these tests separately. This process

ensures that the deployed application meets the expected quality standards in terms of both functionality and performance

5. Orchestration

Google Cloud Run, the infrastructure where business services in accounting system applications run, has several orchestrator features that are functionally similar to those in Kubernetes. Users are able to define the minimum and maximum number of instances needed for scaling purposes. In the Accounting System application, you can adjust and evaluate these settings over time. One limitation of Cloud Run is the lack of a scheduling feature, which would allow cloud runs to be executed at specific times. These capabilities are typically utilized by services that perform processing at specific times. Users can utilize Google Cloud Scheduler and Google Cloud Function to incorporate these capabilities into Cloud Run

Discussion

After analyzing the architecture of PT XYZ's Accounting System, we reviewed to determine if PT XYZ's design aligns with the principles outlined by CNCF and the 12 Factor App. Overall, PT XYZ's architecture meets most of the principles of Cloud Native architecture, with some areas identified for potential improvement.

The architecture used by PT XYZ indicates that all business services built on the Accounting system application share the same database schema. This is not in accordance with the seventh principle, which is data isolation. In data isolation, each service manages its data. Therefore, one service cannot access data belonging to another service directly; it must go through the API provided by the service.

The next point is about the first principle, which is the codebase. According to this principle, each service should be kept in a separate code repository. Currently, all services are stored in the same code repository, but this could create difficulties in the future as the number of services grows.

All evaluations for this research have been submitted to PT XYZ as part of the future application development process. PT XYZ has received the evaluation as material for its upcoming development roadmap.

E. Conclusion

Implementing the PT XYZ accounting system application's microservice architecture needs to be accompanied by the appropriate underlying infrastructure. Therefore, it is essential to incorporate a cloud-native architecture that includes elements such as containerization, dynamic management, microservices, automation, and orchestration.

PT XYZ has successfully incorporated cloud-native architecture and most twelve-factor app principles. They have implemented containerization using Google Cloud Run, dynamic management using AWS and GCP services, and automated CI/CD processes through Jenkins. This shows a strong integration of cloud technology in supporting their accounting system operations.

However, several areas require improvement to achieve full optimization of cloud-native architecture. One of them is the principle of data isolation, which has not been fully implemented, where all business services share the same database schema. In addition, the use of separate code repositories in each service needs to

be reviewed, as it may pose development difficulties in the future as the number of developed services increases

F. References

[1] Gartner, "Gartner Forecasts Worldwide Public Cloud End-User Spending to Reach $679 Billion in 2024." Accessed: Jun. 05, 2024. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/11-13-2023-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-679-billion-in-20240

[2] Ravenry, "SaaS Wave in Indonesia," 2020. Accessed: Mar. 06, 2024. [Online]. Available: https://theravenry.com/wp-content/uploads/2020/08/SaaS-Wave-in-Indonesia-Booklet.pdf

[3] M. Wu et al., "On the Way to Microservices: Exploring Problems and Solutions from Online Q&A Community," in 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), Mar. 2022, pp. 432–443. doi: 10.1109/SANER53432.2022.00058.

[4] P. Mangwani, N. Mangwani, and S. Motwani, "Evaluation of a Multitenant SaaS Using Monolithic and Microservice Architectures," SN Comput Sci, vol. 4, no. 2, Mar. 2023, doi: 10.1007/s42979-022-01610-2.

[5] P. H. Nguyen, H. Song, F. Chauvel, R. Muller, S. Boyar, and E. Levin, "Using Microservices for Non-Intrusive Customization of Multi-Tenant SaaS," in Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, in ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, pp. 905–915. doi: 10.1145/3338906.3340452.

[6] F. and N. P. H. Song Hui and Chauvel, "Using Microservices to Customize Multi-tenant Software-as-a-Service," in Microservices: Science and Engineering, N. and D. S. and L. P. and M. M. and R. V. and S. A. Bucchiarone Antonio and Dragoni, Ed., Cham: Springer International Publishing, 2020, pp. 299–331. doi: 10.1007/978-3-030-31646-4_12.

[7] P. Mangwani and V. Tokekar, "Container Based Scalability and Performance Analysis of Multitenant SaaS Applications," in 2022 13th International Conference on Computing Communication and Networking Technologies, ICCCNT 2022, Institute of Electrical and Electronics Engineers Inc., 2022. doi: 10.1109/ICCCNT54827.2022.9984214.

[8] M. I. Josélyne, D. Tuheirwe-Mukasa, B. Kanagwa, and J. Balikuddembe, "Partitioning microservices: A domain engineering approach," in Proceedings - International Conference on Software Engineering, IEEE Computer Society, May 2018, pp. 43–49. doi: 10.1145/3195528.3195535.

[9] V. M. Niño-Martínez, J. O. Ocharán-Hernández, X. Limón, and J. C. Pérez-Arriaga, "A Microservice Deployment Guide," Programming and Computer Software, vol. 48, no. 8, pp. 632–645, Dec. 2022, doi: 10.1134/S0361768822080151.

[10] P. Di Francesco, P. Lago, and I. Malavolta, "Migrating Towards Microservice Architectures: An Industrial Survey," in Proceedings - 2018 IEEE 15th International Conference on Software Architecture, ICSA 2018, Institute of

Electrical and Electronics Engineers Inc., Jul. 2018, pp. 29–38. doi: 10.1109/ICSA.2018.00012.
[11] S. G. Haugeland, P. H. Nguyen, H. Song, and F. Chauvel, "Migrating Monoliths to Microservices-based Customizable Multi-tenant Cloud-native Apps," in 2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Sep. 2021, pp. 170–177. doi: 10.1109/SEAA53835.2021.00030.
[12] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation," IEEE Cloud Computing, vol. 4, no. 5, pp. 22–32, 2017, doi: 10.1109/MCC.2017.4250931.
[13] A. Henry and Y. Ridene, "Migrating to Microservices," in Microservices, Cham: Springer International Publishing, 2020, pp. 45–72. doi: 10.1007/978-3-030-31646-4_3.
[14] S. Wells, Enabling Microservice Success: Managing Technical, Organizational and Cultural Challenges, Early Release. O'Reilly Media, Inc., 2024.
[15] Y. Wang, H. Kadiyala, and J. Rubin, "Promises and challenges of microservices: an exploratory study," Empir Softw Eng, vol. 26, no. 4, Jul. 2021, doi: 10.1007/s10664-020-09910-y.
[16] E. Jiang, A. McCright, J. Alcorn, D. Chan, and A. Nottingham, Practical Cloud-Native Java Development with MicroProfile. Packt Publishing, 2021.
[17] T. Vitale, Cloud Native Spring in Action. Manning Publications, 2023.
[18] G. Toffetti, S. Brunner, M. Blöchlinger, J. Spillner, and T. M. Bohnert, "Self-managing cloud-native applications: Design, implementation, and experience," Future Generation Computer Systems, vol. 72, pp. 165–179, Jul. 2017, doi: 10.1016/j.future.2016.09.002.
[19] P. Reznik, J. Dobson, and M. Gienow, Cloud Native Transformation. O'Reilly Media, Inc., 2019.
[20] B. Scholl, T. Swanson, and P. Jausovec, Cloud Native. O'Reilly Media, Inc., 2019.
[21] Z. Lyu, H. Wei, X. Bai, and C. Lian, "Microservice-Based Architecture for an Energy Management System," IEEE Syst J, vol. 14, no. 4, pp. 5061–5072, Dec. 2020, doi: 10.1109/JSYST.2020.2981095.
[22] V. Kornuta, E. Sobotnyk, I. M. Katamai, and Y. Katamai, "Using Microservice Architecture for High-Load Information Systems on the Example of MedicinePlanner Service," in 2022 12th International Conference on Advanced Computer Information Technologies, ACIT 2022, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 437–442. doi: 10.1109/ACIT54803.2022.9913185.
[23] T. Chen and H. Suo, "Design and Practice of DevOps Platform via Cloud Native Technology," in 2022 IEEE 13th International Conference on Software Engineering and Service Science (ICSESS), IEEE, Oct. 2022, pp. 297–300. doi: 10.1109/ICSESS54813.2022.9930226.
[24] D. Bharadwaj, S. Premananda B., D. of Electronics, T. Department, of Electronics, and Telecommunication, "Transition of Cloud Computing from Traditional Applications to the Cloud Native Approach," in 2022 IEEE North

Karnataka Subsection Flagship International Conference (NKCon), IEEE, 2022

[25] A. Rosilier, M. Demir, and J. Prevost, “Automated Consulting for Cloud Native Architectures.” IEEE, Jun. 2022.