ComTech: Computer, Mathematics and Engineering Applications, 11(2), December 2020, 89-95 P-ISSN: 2087-1244
DOI: 10.21512/comtech.v11i2.6453 E-ISSN: 2476-907X

Implementation of Microservices Architecture on E-Commerce Web Service

Juan Andrew Suthendra[1]* and Magdalena Ariance Ineke Pakereng[2]

[1,2]Department of Informatics Engineering, Faculty of Information Technology, Satya Wacana Christian University
Jln. Diponegoro 52-60, Salatiga 50711, Indonesia
[1]juanandrew9912@gmail.com; [2]ineke.pakereng@uksw.edu

Received: 15th May 2020/ **Revised:** 13th August 2020/ **Accepted:** 18th August 2020

How to Cite: Suthendra, J. A., & Pakereng, M. A. I. (2020). Implementation of Microservices Architecture on E-Commerce Web Service. *ComTech: Computer, Mathematics and Engineering Applications, 11*(2), 89-95. https://doi.org/10.21512/comtech.v11i2.6453

Abstract **-** The research aimed to make e-commerce web services using a microservices architecture. Web service was built using Representational State Transfer Protocol (REST) with Hypertext Transfer Protocol (HTTP) method and JavaScript Object Notation (JSON) response format. Meanwhile, the microservices architecture was developed using Domain-driven Design (DDD) approach. The research began by analyzing e-commerce business processes and was modeled using Unified Modeling Language (UML) based on business process analysis. Next, the bounded context was used to make a small responsible service for a function. The Programming language used to make the system was Go programming language with Go-kit tool and apply database-per-service pattern for data management. The system also applied the concept of containerization using Docker as the container platform and using API Gateway to manage each endpoint. Last, the evaluation process was carried out using the Postman application by testing each endpoint based on the white-box testing method. Based on the results of the evaluation process, the e-commerce web service can work as expected. The results also show that the system has a high level of resilience. It means that the system has a low level of dependencies between services and adapts to future changes.

Keywords: microservices architecture, e-commerce web service

I. INTRODUCTION

E-commerce is used to sell or buy products on online services or over the Internet. Nowadays, e-commerce is showing significant growth, thanks to the help of a cashless payment trend and the global digital revolution. Asia leads in the first place in the aspect of digitalization, including e-commerce. Indonesia recorded more than 20% sales growth on e-commerce transactions in 2017 (Kinda, 2019).

According to Khan (2016), e-commerce has benefits for customers such as time-saving, convenience, product variations, comfort, and ease of getting the product information. Meanwhile, for the sellers, the benefits are reducing the operational cost, the maintenance cost, and the procurement cost, increasing revenue, developing the company image, and raising customer loyalty. To support the significant growth of e-commerce and maximize its benefits, e-commerce must provide services that have high availability and are easy to maintain and develop.

Web service is a standardized way to propagate communication between client and server. There are two common protocols used in making web services that are Representational State Transfer Protocol (REST) and Simple Object Access Protocol (SOAP) (Tihomirovs & Grabis, 2016). REST can use JavaScript Object Notation (JSON) or Extensible Markup Language (XML) as the data format. Meanwhile, SOAP can only use XML as the data format. Advantages of REST over SOAP are better performance, more simplicity, fast execution, greater scalability, more loose coupling, and lower memory consumption. On the other hand, SOAP is more secure and reliable. SOAP is best suited for long-haul working projects due to having more focus on security and reliability, such as banking, financial, and telecommunication services. Then, REST is for projects with big scale because of focusing more on simplicity and performance, such as web chats and mobile services (Soni & Ranga, 2019).

In the past, a system is usually built under monolithic architecture. Monolithic architecture is described as a single-tiered software application in which the user interface and data access code are combined into a single program from a single platform (Venugopal, 2017). Over time, the system will continue to experience changes due to changes or growth in the business processes that cause the application to become more complex and bigger. Monolithic architecture weakness is the ability to adapt when the system changes its requirement, especially in managing its code complexity and code maintainability. It will be a problem in the distribution process due to its high dependency levels (Munawar & Hodijah, 2018). If one part of the code is changed, it will affect the other parts of the code. So, the other parts of the code must also be changed.

To solve that problem, the system will be built using a microservices architecture. Microservices are small

*Corresponding Author

autonomous services that work together (Newman, 2015). Microservices split a large service into smaller services. Each service has duties and operates independently. Hence, it causes a low level of dependencies between services (loose coupling). The low level of dependencies between services makes it easier for the system to adapt to changes (Munawar & Hodijah, 2018). It will also be easier for the developer to modify some services and deploy them without changing the whole system. One way to make the services has a low dependency level is by reducing the communication between services (Newman, 2015). Communication between services is only done through communication between Application Programming Interface (API) (Richardson, 2018). Splitting the services also makes microservices architecture have a faster response time compared to monolithic architecture (Budi, 2018). Another advantage of microservices architecture lies in system resilience. System resilience is the ability to withstand error or infrastructure damage. The system can still work even if there are interrupted services (Suryotrisongko, 2017). However, to guarantee the resilience of the system, microservices must be well monitored. All kinds of logging and operational data must be collected consciously and consistently (Singhal, Sakthivel, & Raj, 2019). The logged data can help developers to find problems occurring in the system.

Domain-Driven Design (DDD) is an approach to develop a microservices infrastructure. It focuses on the domain, which includes concepts, relationships between domains, and existing business processes (Steinegger, Giessler, Hippchen, & Abeck, 2017). It uses bounded context to identify services on the microservice architecture. Business processes should be grouped according to their function. Each group will become the bounded context of the system (Newman, 2015). Then, bounded context turns into small services in the microservices architecture.

From the explanation mentioned, the goal of the research is to make an e-commerce web service using a microservices architecture. The system is expected to have a high level of resilience and adapt to changes in the future.

II. METHODS

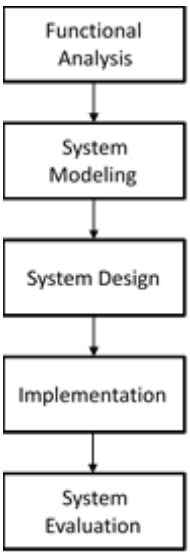

Figure 1 Research Method

The used research method can be seen in Figure 1. The first step is functional analysis. It is the process of identifying and analyzing business processes. Then, the data collection is conducted to obtain the features that will be provided in the system. Each feature has a function and an actor who can access it.

The second step is system modeling. The system is modeled according to the result of functional analysis. It is modeled using the Unified Modeling Language (UML). The result shows the interaction between actors and the system in the form of a use case diagram.

The third step is system design. This stage is carried out by designing the microservices. The results describe the used technology and the system configuration. The system is built using the Go programming language with the Go-Kit tool. Go has a better server-side in Input/Output (I/O) performance than Java, Node, and PHP (Peabody, n.d). There is another toolkit that can be used to build microservices architecture in Go, called Go-Micro. Both Go-Kit and Go-Micro provides features like service discovery, load balancing, and others. The difference between Go-Kit and Go-Micro lies in the opinion and expectation about the infrastructure and architecture. Next, the database is designed using a database-per-service pattern. This pattern allows selecting a suitable database for each service, maintains the data needed by related services, and prevents other services to access the database directly. The data transaction between services can only occur through API communications. The system also applies the containerization concept using Docker as the container platform. The concept of containerization provides advantages in program design, monitoring, and implementation (Singhal *et al.*, 2019).

Microservices should be packaged in the container image. Then, the scaling is done based on changing the container number instance, and the performance will be much faster than running on a regular Virtual Machine (VM) (Venugopal, 2017). API Gateway is used to manage each endpoint. It is an API management tool between a client and a system. It acts as the only way to access the system and helps in the process of securing, configuring, and encapsulating the internal structure of the system (Zhao, Jing, & Jiang, 2018).

The fourth step is implementation. In this stage, the system is made following the technology that has been determined. The database is the first part to be created and implemented using the database-per-service pattern. Each service has a database. After creating databases, the web service is coded. Go-Kit tools are laid out in three layers: transport layer, endpoint layer, and service layer. Request enters the system in the transport layer and flows down to the service layer. Meanwhile, responses take the reverse course. The transport layer determines the used transport protocol in the system. This system uses HTTP as the transport protocol. The endpoint layer is the system controller. This layer controls the action flow of the system. The service layer holds the business process of the system. After the code is done, all of the services need to be registered into API Gateway. Then, the API Gateway is configured. The last step is to package the system in the Docker container.

The final step is system evaluation. The system is evaluated to ensure that the system can work as expected and have a good resilience level. Every system endpoint is tested using the white box testing method. Then, test results are compared with expectations. If the results and expectations of the test are similar, the endpoint will be

declared successful. If it is not, the endpoint will be fixed and tested again. If all endpoints work as expected, the next step is to test the resilience of the system. The test simulates system failure in real scenarios. Some service nodes are shut down, and the nodes that are still functioning will be tested. The system will pass the resilience test if the alive service nodes can perform as expected. The system will be successful if it passes both tests.

III. RESULTS AND DISCUSSIONS

The functional analysis for produced data that are related to the features is needed in the system. Based on the functional analysis result, there are two actors in the system: customer and merchant. The customer is a member of the system that has a role as a buyer. Meanwhile, the merchant has a role as a seller. The covered business process by the system is the transaction between customer and merchant, including the shipping process. However, the payment process is not covered in the system. The payment system uses a third-party payment processor. The payment process uses a third-party payment processor to ensure the security and automation of the payment process. Table 1 shows the result of functional analysis. It shows the activity of each actor in the system.

Based on the data in Table 1, a use case diagram is designed. The use case diagram illustrates the interaction between the system and the actors. Figure 2 shows a use case diagram regarding the interaction between the actors and the system. In this system, customer can manage their account, order product, and view a list of products and shipping information. Moreover, the merchant can manage their account, shipping information, orders, and product information.

After creating the use case diagram, the researchers determine the bounded context in the system based on the business process. It is a logical constraint that has the responsibility for implementing a business process in the system. The DDD approach uses bounded context to determine services in the microservices architecture. Table 2 shows the bounded contexts in the system. The system has four services: account, inventory, order, and shipping.

Table 1 Result of Functional Analysis

Actor	Activity
Customer & Merchant	Create an account and manage account information
Customer	Order products
	See a list of products
	See shipping information
Merchant	Sell products
	Manage products information
	Manage order information
	Manage delivery information

Table 2 Bounded Context

Bounded Context	Description
Account	This service is responsible for managing account data, registering a new account, and handling the users' login process.
Inventory	This service is responsible for managing product information.
Order	This service is responsible for managing order information.
Shipping	This service is responsible for managing delivery information.

Account service is responsible for managing customer and merchant account. A merchant account can only be created if the user already has a customer account. Customer account management manages personal data and addresses. The stored personal data by the system are account ID, email, password, name, telephone number, gender, date of birth, and account creation date. An account can store multiple shipping addresses. Moreover, merchant account management saves the store data. The system keeps merchant data such as merchant ID, account ID, merchant name, address, and credit.

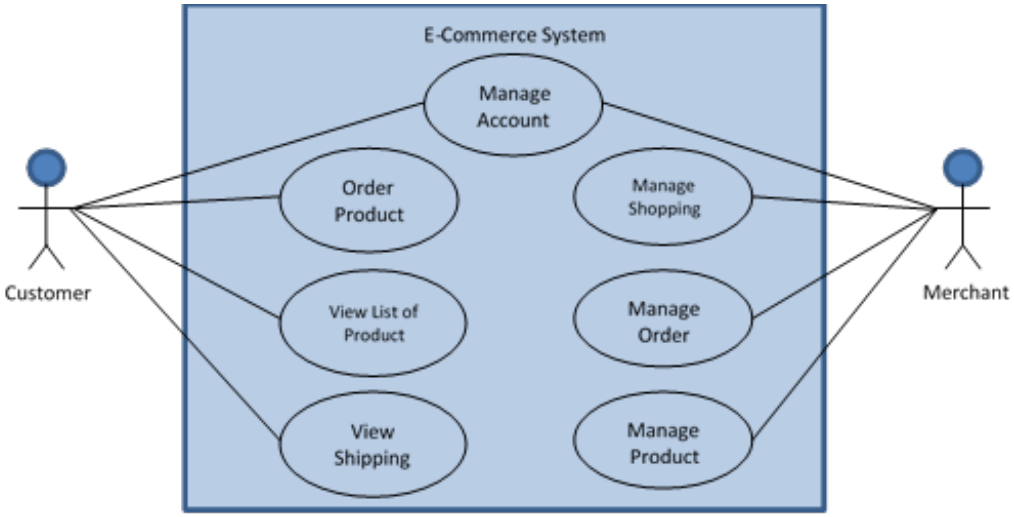

Figure 2 Use Case Diagram

Inventory service is responsible for managing product data from each merchant. Products data that are stored by the system are product ID, merchant ID, product name, stock, description, and price. Next, the order service manages the orders. Customers can place orders. Meanwhile, merchants can receive and process the orders. In managing the order data, data will be placed in two different tables. The first table stores order data such as order ID, customer ID, merchant ID, shipping address, order status, order date, and last update date. Meanwhile, the second table saves the details of each order. The data are order ID, product ID, product amount, unit price, and total price. Then, the shipping service is responsible for managing shipping data. The data are shipping ID order ID, courier, shipping date, and tracking ID.

Table 3 shows the endpoints and their function in the system. Web service is built using REST architecture. REST architecture consists of five Hypertext Transfer Protocol (HTTP) methods that can be used. These methods are POST, GET, PATCH, DELETE, and PUT. POST creates a new resource. GET retrieves a specific resource. PATCH is used to update a specific resource. DELETE removes a specific resource, and PUT replaces all specific resources.

Each endpoint is registered to an API Gateway using Kong. It is configured by adding a circuit breaker. The circuit breaker will limit the total requests that can be made to the system in a period. When the number of requests has reached a specified limit, the circuit breaker will activate, prevent the request from getting to endpoint services, and return an error message. This method can protect the system from the chaos that can occur when a service fails to fulfill the user request.

Figure 3 is the design of microservices architecture. The system is built using REST with HTTP. The response given from every request is sent in the form of JSON. The used database by all services is PostgreSQL. Then, the system is equipped with a security layer to protect data and communication processes between users and the system. Password is protected using the BCrypt hashing algorithm. BCrypt is an encryption algorithm based on the blowfish encryption algorithm. It meets three security criteria that are believed to be sufficient to protect a system. It also has the second preimage-resistance, sufficient salt capacity (in cryptography) to protect from pre-computational attack, and adaptable cost (Provos & Mazieres, 1999).

JSON Web Token (JWT) is used when users want to access the endpoints in the system. JWT will ensure that data will only be accessed by certain accounts by comparing the token claims.

Next, the evaluation of the system is done using the white-box testing method. Table 4 shows the result of the system evaluation. The code column is the endpoint that will be tested. The code refers to Table 3. The examination column is the action given to the system. Meanwhile, the expectation column is the expected result when the users access the endpoint. The reality column is the test result obtained. Then, the result column shows whether the action is successfully conducted or not.

Table 3 The Created Endpoints

Service	Code	Endpoints	Method	Function
Account	A1	/signup	POST	Create customer account
	A2	/signin	POST	Sign in to the existing account
	A3	/profile/{id}	GET	Show customer data
	A4	/address/{user_id}	GET	Show list of customer addresses
	A5	/address/id/{id}	GET	Show information about selected address
	A6	/address	POST	Add new customer address
	A7	/address	PATCH	Change selected customer address
	A8	/address/{id}	DELETE	Delete selected customer address
	A9	/merchant/{user_id}	GET	Show merchant data
	A10	/merchant	POST	Create a merchant account
	A11	/merchant	PATCH	Change merchant data
Inventory	I1	/product	POST	Add new product
	I2	/product	PATCH	Change the selected product data
	I3	/product/{merchant_id}	GET	Show list of product sold by a merchant
	I4	/product/id/{id}	GET	Show product information
	I5	/product/stock	PATCH	Change the stock amount of a product
Order	O1	/order	POST	Order products
	O2	/order/status	PATCH	Change order status
	O3	/order/{user_id}	GET	Show list of orders made by an account
	O4	/order/{merchant_id}	GET	Show list of orders made to a merchant
	O5	/order/id/{id}	GET	Show data about an order
Shipping	S1	/shipping	POST	Create new shipping information
	S2	/shipping/{order_id}	GET	Show shipping information of an order

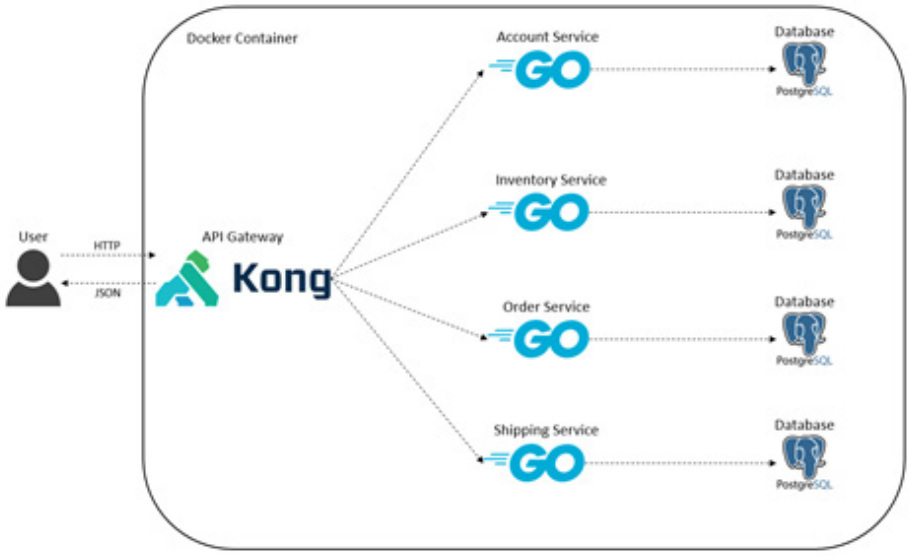

Figure 3 Microservice Architecture

Table 4 The Result of System Evaluation

Code	Examination	Expectation	Reality	Result
A1	Users create a new customer account and fill in the required information.	A new account is created, the password is encrypted, and all information is saved on the database.	A new account is created, the password is encrypted, and all information is saved on the database.	Success
	Users create a new customer account and do not fill in the required information.	The system returns an error message.	The system returns an error message.	Success
A2	Users input the correct email and password combination.	Users log in to the system.	Users log in to the system.	Success
	Users input incorrect email and password combination.	System returns an error message indicating the incorrect combination of email and password.	System returns an error message indicating the incorrect combination of email and password.	Success
A3	Customers see their personal information. The account ID is sent to the system.	The system receives their account ID and shows users' information.	The system receives their account ID and shows users' information.	Success
A4	Customers see their address list. The account ID is sent to the system.	The system receives their account ID and shows the saved address on their account.	The system receives their account id and shows the saved address on their account.	Success
A5	Customers see their address information. The address ID is sent to the system.	The system receives address ID and shows address information.	The system receives address ID and shows address information.	Success
A6	Customers register a new address to their account.	The system saves the address information to the database.	The system saves the address information to the database.	Success
A7	Customers select one address and change the address information.	System changes the selected address information	System changes the selected address information	Success
A8	Customers select one address and delete it.	The selected address is deleted from the database.	The selected address is deleted from the database.	Success
A9	Customers have a merchant account and want to see their merchant information.	The system shows the merchant information.	The system shows the merchant information.	Success
	Customers do not have a merchant account. They want to see their merchant information.	The system returns an error message.	The system returns an error message.	Success

A10	Users create a new merchant account.	A new merchant account is created, and all information is saved on the database.	A new merchant account is created, and all information is saved on the database.	Success
A11	Merchant changes their information.	Merchant's information is changed, and all information is saved on the database.	Merchant's information is changed, and all information is saved on the database.	Success
I1	Merchant adds a new product to the showcase.	Product is added to the merchant's showcase.	Product is added to the merchant's showcase.	Success
I2	Merchant selects a product and changes its information.	The system changes the selected product information.	The system changes the selected product information.	Success
I3	Customers see merchant showcase	The system shows the selected merchant showcase.	The system shows the selected merchant showcase.	Success
I4	Customers select a product sold by the merchant.	The system shows the selected product information.	The system shows the selected product information.	Success
I5	Merchant changes the stock amount of a product.	The system changes the amount of stock on the selected product.	The system changes the amount of stock on the selected product.	Success
O1	The customer makes an order.	The order information is saved in the database. The amount of stock of a product ordered is reduced.	The order information is saved in the database. The amount of stock of a product ordered is reduced.	Success
O2	Order status is changed.	The system changes the status of an order. If the order status is canceled, the quantity of an ordered product will return to the product stock amount.	The system changes the status of an order. If the order status is canceled, the quantity of an ordered product will return to the product stock amount.	Success
O3	Customers see their orders.	The system shows a list of orders made by a customer.	The system shows a list of orders made by a customer.	Success
O4	Merchants see the order made by customers to their products.	The system shows a list of orders made by a customer to a merchant.	The system shows a list of orders made by a customer to a merchant.	Success
O5	Customers or merchants see order details.	The system shows the order details.	The system shows the order details	Success
S1	Merchant adds shipping information to an order.	Shipping information of an order is added to the database.	Shipping information of an order is added to the database.	Success
S2	Customers see the shipping information of an order.	The system shows the shipping information of an order.	The system shows the shipping information of an order.	Success

Last, the evaluation process of the system resilience is carried out by simulating the system failure. Several service nodes are shut down randomly. Service nodes that are still functioning are tested. The result of the test shows the service nodes that do not experience the interference can work normally.

IV. CONCLUSIONS

Based on the results of the evaluation process, the e-commerce web service can work as expected. The evaluation results also show that the system has a high level of resilience. It means that the system has a low level of dependencies between services and adapts to future changes. Docker can help developers to develop a system. It helps the developers in the development, implementation, and deployment process. Docker will package all of the used libraries and dependencies in the system into a package. It will simplify the process of deploying and distributing the system. In making a system using microservices architecture, the whole system is broken down into many smaller services. Then, several factors must be considered in splitting the service, such as service dependencies and service communication. Microservices must be well designed. Otherwise, the development of the system will be difficult.

The advantages of using microservices architecture in developing a system are the flexibility and maintainability of the system. The system performance can also be maximized because it can be built using different programming languages and databases. Changes and improvements in service will not affect the works of other services as long as the service is not dependent on each other. It is essential because business processes will continue to grow, and the system must adapt to the changes.

For future research, some parts can be improved. First, apply the concept of Event Sourcing. This concept will help the data consistency and record system activities. Second, use the Command Query Responsibility Segregation (CQRS) pattern in performing a query that involves many services. Third, use newer and more secure password encryption techniques such as Argon2.

REFERENCES

Budi, C. S. (2018). *Implementasi arsitektur microservices pada backend comrades* (Doctoral dissertation). Universitas Komputer Indonesia

Khan, A. G. (2016). Electronic commerce: A study on benefits and challenges in an emerging economy. *Global Journal of Management and Business Research: B Economics and Commerce, 16*(1), 19-22.

Kinda, M. T. (2019). *E-commerce as a potential new engine for growth in Asia*. International Monetary Fund.

Munawar, G., & Hodijah, A. (2018). Analisis model arsitektur microservice pada sistem informasi DPLK. *Sinkron: Jurnal dan Penelitian Teknik Informatika*, *3*(1), 232-238.

Newman, S. (2015). *Building microservices: Designing fine-grained systems*. USA: O'Reilly Media, Inc.

Peabody, B. (n.d). *Server-side I/O performance: Node vs. PHP vs. Java vs. Go*. Retrieved August 13th 2020 from https://www.toptal.com/back-end/server-side-io-performance-node-php-java-go

Provos, N., & Mazieres, D. (1999). A future-adaptable password scheme. In *USENIX Annual Technical Conference, FREENIX Track* (pp. 81-91).

Richardson, C. (2018). *Microservices patterns*. Manning.

Singhal, N., Sakthivel, U., & Raj, P. (2019). Selection mechanism of micro-services orchestration vs. choreography. *International Journal of Web & Semantic Technology (IJWesT)*, *10*(1), 1-13.

Soni, A., & Ranga, V. (2019). API features individualizing of web services: REST and SOAP. *International Journal of Innovative Technology and Exploring Engineering*, *8*(9S), 664-671.

Steinegger, R. H., Giessler, P., Hippchen, B., & Abeck, S. (2017). Overview of a domain-driven design approach to build microservice-based applications. In *The Third International Conference on Advances and Trends in Software Engineering (SOFTENG 2017)*.

Suryotrisongko, H. (2017). Arsitektur microservice untuk resiliensi sistem informasi. *Jurnal SISFO: Inspirasi Profesional Sistem Informasi, 6*(2), 235-250.

Tihomirovs, J., & Grabis, J. (2016). Comparison of SOAP and REST based web services using software evaluation metrics. *Information Technology and Management Science*, *19*(1), 92-97.

Venugopal, M. V. L. N. (2017). Containerized microservices architecture. *International Journal of Engineering And Computer Science*, *6*(11), 23199-23208.

Zhao, J. T., Jing, S. Y., & Jiang, L. Z. (2018). Management of API gateway based on micro-service architecture. *Journal of Physics: Conference Series, 1087*, 1-8.