Design of Information System Architecture of Garment Enterprises Based on Microservices

Weilun Tang\textsuperscript{1}, Li Wang\textsuperscript{1,a} and Guangtao Xue\textsuperscript{1}

\textsuperscript{1}Department of Computer Science and Engineering, Shanghai Jiao Tong University, China

\textsuperscript{a} Corresponding author: wang-li@cs.sjtu.edu.cn

Abstract. The microservices architecture has been favored by system designers and has gradually become the mainstream solution in enterprise-level application architecture design since its introduction. This article aims at the urgent need for information transformation and upgrading faced by the garment industry, and designs a system architecture based on microservices to solve the two problems it faces: the first is how to design an enterprise resource planning (ERP) system that improves the efficiency of enterprise management which is ability to develop and apply quickly; the second is how to upgrade module-level functions on the original information management system smoothly. This paper decomposes the business process into microservices and designs an asynchronous communication mechanism between microservices, enhances system scalability and robustness. This paper uses Docker so that the enterprise's original ERP system can be compatible with the new system and gradually upgrade its function modules.

1. Introduction

As a traditional manufacturing industry, the garment industry is extremely sensitive to fashion trends and needs to respond more quickly and flexibly to changes in social trends. Therefore, transformation and upgrading are urgently needed. The system is based on cooperation with a clothing company to solve the two major technical challenges.

First, the company's communication with upstream customers and downstream factories, as well as internal departments of the company, relies on offline telephones and mails, which is inefficient and leads to serious waste of production materials. At the same time, the types and quantity of clothing styles are huge, and the production steps are complicated, so that there are many departments in need of coordination. The company's business department cannot timely feedback information to each participant. Once the entire production process is adjusted, the participants can only be re-notified and coordinated by mail or telephone, resulting in inefficient work.

The second is how the company's original ERP system is compatible with the new system, and gradually upgrade.

At present, an ERP system is being run within the company, but it cannot meet the needs of enterprise development. Therefore, the new architecture needs to be compatible with the original ERP system, so that the original system works normally in the new operating environment, and can gradually replace or increase its functions without affecting the normal production of the company, causing unnecessary losses.

In order to solve these two problems, an ERP system architecture based on microservices [1] was proposed. The microservices decomposes the business process into microservices according domain
and use the asynchronous communication mechanism of the message bus to deliver messages, which supports the real-time information interaction between the company, the customer and the factory, which enhances the operational efficiency of each department. The scalability and compatibility of microservices, as well as the Docker container’s isolation of the infrastructure [2], makes the original system can be compatible with the new system framework, and can achieve functional modules partial replacement.

2. Related Work

N. Dragoni [3] elaborates on the development of microservices and introduces its commonly used components and the challenges faced by microservices. M. Richards [4] compares the microservice architecture with the traditional SOA architecture, pointing out that the microservice architecture is an extension of the SOA architecture, but it is more focused on the componentization, agile development and DevOps practices of small and lightweight services.

At present, various microservice frameworks such as Spring Cloud and Dubbo have been proposed. With the help of these technologies, microservices have had many successful applications. N. Dragoni [5] successfully reconstructed the mission-critical system of Denmark's largest bank Dansk Bank from a single system to a micro-service architecture and 6. A. Bucchiarone [6] traced the system, which is considered to greatly improve the scalability of the system. D. Guo [7] proposed a new cloud-based PaaS platform based on lightweight container technology and microservices architecture. R. Ouyang [8] designed a data service framework based on microservices.

3. Methodology

Microservices is a new system architecture design style proposed by Martin Fowler in his blog on Microservices in March 2014[9]. This architecture decomposes traditional single applications into multiple highly coupling small services by domain, each of which is developed, tested, deployed and maintained in a separate environment. The services communicate through lightweight communication mechanisms (usually based on RESTful and JSON), which greatly reduces coupling between services.

3.1. Framework design

To solve the problem of low management efficiency, one is to decompose the entire business process of the company and make it become independent services; the second is to use the advantage of computer network to directly transfer messages between services and avoid excessive manual operations.

In order to integrate the company's business with the original ERP system, this paper divides “planning service”, “material service”, “purchasing service”, “warehouse service”, “production service” and “sales service” of the company. Each service communicates through "messages" and automatically pushes the business.

To solve the compatibility problem of the original ERP system, the Docker virtual environment is used to create the isolate environment required by the original system in the new system environment [10], so as to ensure the normal operation of the original system. On this basis, using the good scalability of the microservices architecture itself, a new service is designed for each original ERP system module, so as to gradually replace the original ERP system without one-time overall replacement.
As shown in Figure 1, the microservices architecture is mainly divided into service discovery, load balancing, service gateway, message bus, authentication system, and the six new microservices and original ERP systems with Docker container mentioned above.

3.2 Service discovery and request forwarding

Service discovery and request forwarding include service discovery, load balancing, service gateway and authentication system.

Service discovery is the registration center for distributed services. Each service instance automatically registers its own service name, IP address, and port number after it is started. It is sent to the service discovery module in JSON format through restful request. When registering, the same name will be considered as different instances of the same service for allocation of the load balancing algorithm. After the service is successfully started, a heartbeat connection is continuously sent to the service discovery every second to indicate its health status. The JSON format as follows:

```
{
    "name": "service_plan",
    "ip": "127.0.0.1",
    "port": 8080,
    "status": "start",
    "timestamp": 1535511448
}
```

When the service finds that the heartbeat connection has not been received for three consecutive seconds or receives an abnormal state heartbeat connection, the service instance is removed from the usable list and sent to the protected area. After the service resumes normal heartbeat, it is re-inserted into the usable list.

The service gateway cooperates with the service discover undertake the task of requesting forwarding, which caches the address list of the service instances available in the service discovery, reducing its access pressure. The requests of the system users are sent to the service gateway uniformly, and the requested service name will be carried in the request. The service gateway will query the IP address and port number of service instance or its load balancing from the local cache according to the service name. If not, it will request from the service discovery module, and then forward the request to the address of the real service provider. At the same time, the service gateway also undertakes the task of partial authentication, filtering out illegal tasks to ensure system security.

The system's load balancer uses seven layers of request forwarding. The main function is to obtain all instances of available services from service discovery, and randomly select an instance to forward restful requests to the service instance, thereby reducing the pressure on a single server.

The authentication system is divided into authorization service and access control. The access control is undertaken by the gateway, and the user account, password, role, user group and other information of the system are managed in the authorization service. At the same time, in order to adapt
to the secure access between multiple different services, the system uses OAuth2 technology to achieve SSO (Single Sign On). JWT (JSON Web Token) [11] technology is adopted to achieve it.

JWT is mainly divided into three parts: HEADER, PAYLOAD and VERIFY SIGNATURE. HEADER mainly describes the encryption algorithm and Token type used. PAYLOAD is the body of the message contained in the Token and typically contains the requested permission information such as user, role etc. VERIFY SIGNATURE encodes HEADER and PAYLOAD with Base64Url and encrypts it using the algorithm specified in HEADER (the system uses HS256 algorithm) for access control to secure the Token.

The JWT is issued by the authorization service for each session. Only the request holding the corresponding token can access the corresponding service through the gateway.

3.3 Message bus
The message bus adopts the technology of message middleware, which complies with the AMQP (Advanced Message Queuing Protocol) [12], which provides a unified application layer message service standard protocol. In the AMQP, two models of exchange and message queue are mainly defined for routing and buffering messages. The message producer sends the message to the exchange, which compares the key of the message, and routes the message to the corresponding message queue as theirs key, and then message queue caches the message. The message consumer listens to a message queue, and once a message arrives, it can get the corresponding message. Since messages are cached in the message queue, the use of a message bus to manage communication between services can easily turn synchronous communication into asynchronous communication, greatly reducing the coupling between services and improving scalability.

Take the procurement service and warehouse service as an example, as shown in Figure 2. First, the inbound message queue and the outbound message queue are established in the message bus, and are bound by the keys "inbound" and "outbound" with exchanges. After that, the procurement service sends a “purchasing message” to the switch, which uses the pre-agreed JSON format so that the message content is independent of the language, and carries the routing key value “inbound”. After receiving the message, the exchanges compare the routing key of the message with the bound message queue key, and then routes the message to the inbound message queue and the inbound message queue caches it. The warehouse service listens to the inbound message queue. once a message arrives, it can immediately receive the message and parses the corresponding JSON string. Any new service send messages to the exchange in the same JSON format and carry the "inbound" routing key, the warehouse can receive the message from inbound message queue without affecting the normal operation of other services.
3.4 Docker container

Docker is an open source application container engine that has been very popular in recent years. It shares the operating system with the infrastructure layer and avoids the overhead, so it can make full use of system resources. At the same time, Docker technology isolates applications from the infrastructure and limits the CPU, memory, and other resources used by each container to ensure that programs do not interact with each other [13].

The original ERP system is a typical C/S architecture program, and the front and back ends use TCP directly for communication, so the degree of coupling is very high. Therefore, using Docker technology, the original server is placed inside the container and deployed to the operating environment of the microservices framework, and the client can continue to use the original ERP system without much modification.

3.5 Services

This paper divides six services: “planning service”, “material service”, “purchasing service”, “warehouse service”, “production service” and “sales service”.

The planning service abstracts the entire production process into a process consisting of many sub-activities, creates a template for each activity, defines its start time, duration, and execution progress, and stipulates its association and displays it in the form of a Gantt chart.

The material service is used to manage the basic information of all the materials needed for the company's production for other services to query.

The purchasing service generates a purchasing plan based on the information in the planning service, and then sends the information such as the material details of the purchasing plan as a "message" to the warehouse service.

The warehouse service manages the quantity attributes of all materials (including raw materials, semi-finished products, and finished products), which are directly into the warehouse and outbound operations according to the "message" sent by other services.

Production services and sales services generate plans based on planning services and send corresponding "messages" to the warehouse service.

The traditional procurement, warehouse, production, and sales management systems are independent, and there is no interaction between the systems. It is necessary to input data in each system manually. By using a unified messaging mechanism and using warehouse services as the
“message” receive center, the microservices framework automatically transfers data between services, basically eliminating the need for telephone mail communication and manual input of data, greatly improving efficiency and reducing manual input errors. Probability. Once a service is added, it only needs to use the same messaging mechanism, and basically does not need to modify the original service, which greatly reduces the impact on the existing system.

4. Evaluation

Some services have been online for around 6 months. Previously, the company obtained information about customer-related activities through e-mail and other means, and then arranged the company-related activities for about one to two days’ delay. With the planning service of this framework, with a unified coordination method, all activities can be updated in real time, and all parties can view changes in time, which greatly saves communication costs.

![Figure 3. Efficiency Comparison](image)

Figure 3 shows the basic situation of the company's three Chinese and six Australian business orders before and after the system operation, and its main activities are shown on the X axis. The original manual communication and management method took an average of 158 and 182 days to complete two orders. With this framework, due to a unified production process coordination method, reducing telephone mail communication, and reducing the manual input by means of message delivery, Chinese and Australian orders can be completed in 130 and 151 days respectively, and the production efficiency is increased by about 19% and 17%. Chinese orders are two percentage points higher due to less transportation time.

5. Conclusions

Due to the low degree of informatization of traditional industries or the lack of information at all, the one-time installation of large-scale systems is often far from the actual demand. This paper makes use of the expansion of the microservices architecture, and gradually informatize the company's business module, and let the employees gradually adapt to the new way of working. With the compatibility of Docker, the original system will continue to run.

Acknowledgment

The authors would like to thank the support of National Key R&D Program of China (2017YFC0803700), Shanghai Talent Development Fund, SJTU arts and science inter-project (No.15JCMY08).

References

[1] Y. Zhai, *Spring Cloud Microservices combat*, Beijing: Publishing House of Electronics Industry, (2017)
[2] A. Liu, “Design and Implementation of DevOps System Based on Docker,” Computer Knowledge and Technology, vol 14, No.07, 69-72, (2018)
[3] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," Present and Ulterior Software Engineering, Springer, Cham, 195-216, (2017)
[4] M. Richards, *Microservices vs. service-oriented architecture*, O'Reilly Media, (2015).
[5] N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara, "Microservices: Migration of a mission critical system," arXiv preprint, arXiv:1704.04173, (2017)

[6] A. Bucchiarone, N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara, "From Monolithic to Microservices: An Experience Report from the Banking Domain," IEEE Software, vol 35.3: 50-55 (2018)

[7] D. Guo, W. Wang, G. Zeng, and Z. Wei, "Microservices architecture based cloudware deployment platform for service computing," Service-Oriented System Engineering (SOSE), 2016 IEEE Symposium on. IEEE, (2016)

[8] R. Ouyang, Q. Wang, X. Long, "A data service framework based on microservices," J.Huazhong Univ.of Sci.& Tech.(Natural Science Edition) S1 :126-120 (2016)

[9] M. Fowler, "Microservices," [Online] (2014) https://martinfowler.com/articles/microservices.html. [Accessed: 15-September-2018]

[10] W. Ding, L. Zhang, J. Zhang, Docker container and container cloud, Beijing: The People's Posts and Telecommunications Press, (2016)

[11] M. Jones, J. Bradley, and N. Sakimura. "Json web token (jwt)," No. RFC 7519. (2015)

[12] S. Vinoski, "Advanced message queuing protocol," IEEE Internet Computing, vol 10.6, (2006)

[13] L. Zhou. Spring Cloud and Docker microservices architecture, Beijing: Publishing House of Electronics Industry, (2017)