Journal of Theoretical and Applied Information Technology
15th May 2022. Vol.100. No 9
© 2022 Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

MODEL-BASED RESILIENCE PATTERN ANALYSIS FOR FAULT TOLERANCE IN REACTIVE MICROSERVICE

MUHAMMAD MIRAJ **[1], AHMAD NURUL FAJAR [2]**

[1,2]Information Systems Management Department, BINUS Graduate Program - Master of Information Systems Management, Bina Nusantara University, Jakarta 11480. Indonesia

E-mail: [1]muhammad.miraj@binus.ac.id, [2]afajar@binus.edu

ABSTRACT

In designing the application, it must be robust, meaning that the application design must be able to cope in the event of a failure. The failure here is more of a failure in communication between microservices. Overcoming communication failures between microservices is one of the most difficult problems to solve, especially in distributed systems. In this study, we use a resistance pattern consisting of a circuit breaker, bulkhead pattern, timeout pattern, and retry pattern to test which resistance pattern has a better response time and throughput when there is a failure to communicate between microservices and it can be concluded that the circuit breaker and the timeout pattern has better response time and throughput compared to other resilience patterns.

Keywords: *Microservices, Resilience Pattern, Reactive, Fault Tolerance, Architecture TradeOff Analysis Method*

1 INTRODUCTION

This research was conducted at one of the companies engaged in online travel agents in Indonesia, the company has many microservices and is separated by platform, namely planes, hotels, trains, events, and car rentals. One of the microservices on the aircraft platform, called the flight order manager, has two important functions, namely making insurance orders and making the process of making aircraft orders.

The search process, selecting passengers, schedules, and airlines along with addons such as baggage, food or seats using other microservices, after the process has been completed then the process from the microservice flight order manager runs to form orders and make insurance orders to another microservice called microservice insurance.

When microservice insurance experiences stress, in another sense, it is experiencing a high load, because it does not only communicate with the aircraft platform, but also with the hotel, train, event and car rental platforms. This causes the flight order manager to experience a high load because it does not get a response from the microservice insurance which results in slow consumer transactions and even the transaction fails.

The solution at that time was to temporarily close microservice insurance so that consumers do not buy insurance and microservice flight order managers do not need to make requests because there are no insurance products purchased by consumers. This solution has an impact on the company's purchase of insurance suspended.

Based on these problems the author wants to try to overcome these problems, one way to solve these problems is by using the pattern of resilience [1]. While the resilience pattern itself is a type of service architecture that helps prevent tiered client server communication failures and to maintain functionality in the event of a communication failure in the service. The resilience pattern itself consists of several patterns according to the purpose of the pattern, namely Circuit Breaker Pattern, Bulkhead Pattern, Time Limiter Pattern and Retry Pattern.

The retry pattern concept is quite simple, namely by re-sending the same request as before, while the circuit breaker has a slightly different way of retrying. In the circuit breaker pattern, the client will try to send several different requests to the server. When these requests fail at one time. Then the next request will not be sent again and immediately assume that the request will also fail.

For the time limiter pattern itself, the concept provides a limit to the client not to always send requests. When the request is over-limited, it can immediately reject or create a request queue according to its limit, while the bulkhead pattern is a concept to avoid errors in one part of the system that cause the entire system to shut down.

This study limits the four patterns of resilience to be analyzed, namely the performance of Circuit Breaker, Bulkhead Pattern, Timeout Pattern, and Retry Pattern, evaluated based on response time and throughput so that it is known which of the four patterns has the best performance. Furthermore, an experiment was carried out by conducting a simulation using two microservices, where one of the microservices experienced a system failure, then the microservice would be called by several users simultaneously.

2 LITERATURE REVIEW

.
The literature study was conducted to add references to the theories used in the study as a comparison material in analyzing the model of resilience patterns and the application of the ATAM method as an evaluation method.

2.1 Microservice

Microservice means dividing an application into smaller, interconnected services unlike monolithic applications. Each microservice is a small application that has its own hexagonal architecture consisting of logic and its various adapters (programming languages, etc.).

The microservices architecture is popular for its distributive system style. It describes how to develop an application as a series of small services that are implemented and deployed independently. Each service has processes running and interacting with each other by a lightweight mechanism called an Application Programming Interface (API)[2].

Microservice architectural patterns significantly affect the relationship between applications and databases. Instead of sharing a single database schema with other services, each service has its own database schema. On the one hand, this approach contradicts the idea of an enterprise-wide data model. In addition, it often results in duplication of some data. However, having a per-service database schema is essential if you are to benefit from microservices. Each service has its own database. In addition, services can use the type of database and programming language that best suits their needs.

2.2 Advantages and Disadvantages of microservices

The following are the advantages obtained when using microservices including the following [3]:

- ***Strong Module Boundaries*,** microservices can strengthen the modular structure, which is especially important for larger teams.
- **Independent Deployment**, Simple services are easier to deploy, and because they are autonomous, are less likely to cause system failure when something goes wrong.
- **Technology Diversity**, with mircroservice can combine various programming languages, development frameworks, and data storage technologies.

The following are the disadvantages or prices to pay when using microservices as follows:

- **Distributed**, Distributed systems are more difficult to program, because long-distance calls are slow and always run the risk of failure.
- **Eventual Consistency**, maintaining strong consistency is very difficult for distributed systems, which means everyone must maintain consistency in the end
- **Operational Complexity**, you need a mature operations team to manage

2.3 Reactive System

The term reactive system was introduced by David Harel and Amir Pnueli, and is now generally accepted to designate systems that permanently interact with their environment, and to distinguish them from transformational systems such as compilers [4].

This change occurs because application requirements have changed dramatically in recent years. Just a few years ago large-scale applications had dozens of servers, slow response times, maintenance of gigantic bytes of data offline. Currently, applications run from mobile devices to cloud-based clusters run by multi-core processors. User expectations, response time in milliseconds and 100% uptime. Data is measured in petabytes. Today's software requirements cannot be met by old software architectures.

A coherent approach to the system architecture is urgently needed and believes that all the necessary aspects have been recognized. Reactive Systems must have a system that is Responsive, Resilient, Elastic and Message Based. Systems built as reactive systems are more flexible, loosely coupled, and broad. This makes the system easier to develop and open to change. The system is significantly more failure tolerant and when failure is unavoidable it can handle failure elegantly, avoiding disaster. Reactive Systems are very responsive, can provide users with effective feedback interactively.

2.4 Reactive Manifesto

Prior to 2013, reactive was almost unknown. However, today "reactive" has increased in popularity and is being adopted by more and more companies. This is a direct response to the need for enterprises to design and build applications capable of handling the massively increasing scale and quantity of data. However, this widespread adoption has led to the creation of multiple applications and various "reactive" versions.

In 2013, the Reactive Manifesto was created to do just this. This manifesto was conceived with a view to all the knowledge we have accumulated as an industry in designing and application that is highly reliable and scalable. It is then distilled with this knowledge into a set of required architectural characteristics that will make any application flexible, loose, and elastic. It also carves out a defined vocabulary to enable efficient and clear communication between all participants, including developers, project leaders, architects and CTOs [5].

The reactive manifesto outlines four high-level characteristics of reactive systems: responsive, elastic, resilient, and message driven. While there are all of them, these characteristics are unlike hierarchical levels in standard layered architectures on the contrary, they describe design properties that must be implemented across the technology stack.

2.5 Fault Tolerance

Fault tolerance is a dynamic method used to maintain interconnected relationships, maintainability, and availability in distributed systems. The hardware and software redundancy method are a known fault tolerance technique in distributed systems. Hardware method ensures the addition of multiple hardware devices such as CPU, communication link, memory, and I/O devices while in software fault tolerance method, specific program to resolve the fault. an efficient fault tolerance mechanism helps in detecting faults and if possible recovering from them[6]

Microservices are designed to run in a distributed environment. This distributed environment brings many benefits as well as many challenges. In this paper, the most important characteristics of a distributed system that can affect its behavior negatively will be described. It also provides an overview of best practices on how to address these challenges and the most popular tools for the Java platform that helped follow this writing.

2.6 Resilience Pattern

In making software, it should not just be casual, the software needs to support business processes and customers feel helped in operating the software. If the software is not running in production, it cannot generate value. Productive software, however, must also be correct, reliable, and available.

When it comes to robustness in software design, the main goal is to build robust components that can help errors in their scope, but also the failure of other components that rely on them. Although techniques such as automatic failure or redundancy can make components fault-tolerant, almost every distributed system today. Even simple web applications can contain web servers, databases, firewalls, proxies, load balancers, and cache servers. Moreover, the network infrastructure itself is made up of so many components that there is always a failure going on somewhere[7].

At this writing, the author wants to see four patterns of resistance patterns, namely: Circuit Breaker Pattern, Bulkhead Pattern, Timeout Pattern, Retry Pattern

Circuit Breakers in electronics, circuit breakers are protecting your components from damage due to overload. In software, circuit breaker protects your service from spam while some of it is no longer available due to high load.[7]

Circuit Breaker Pattern described by Martin Fowler. It can be implemented as stateful software that switches between three states: closed (requests can flow freely), open (requests are denied without being sent to a remote resource), and half-open (one request is asked to decide whether to close) the circuit. again).

The Bulkhead Pattern is a type of failure tolerant application design. In a partition architecture, application elements are isolated into batches so that if one fails, the others continue to work. This is selected partition part (bulk) ship hull. If the hull is damaged, only the damaged part is filled with air, so the ship cannot sink[8].

Timeout Pattern is a pattern to set the time limit we wait for the operation to finish called time. If the operation doesn't finish within the time we set, we want to be notified about it with a timeout error. sometimes, this is also referred to as "setting a deadline".

One of the main reasons for this pattern is to ensure that it doesn't keep the user or client waiting indefinitely. Slow service that doesn't provide any feedback can leave users disappointed. Another reason for the pattern of setting a timeout on operations is to make sure we don't hold the resource server indefinitely.

Retry Pattern is Whenever we assume that an unexpected response or no response in this case can be fixed by sending the request again, using the retry pattern can help. This is a very simple pattern where failed requests are retried a configurable number of times in case of failure before being marked as failure [9].

2.7 Architecture TradeOff Analysis Method

The architectural tradeoff analysis method abbreviated as ATAM was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University. Its purpose is to help select alternative architectures for software systems by finding trade-offs and sensitivity points. This method is a risk mitigation process used early in the software development cycle and is most beneficial here because of the minimal cost of architectural changes.

In general, there are 9 steps that can be done with ATAM, of which these 9 steps are categorized into 4 conceptual groups[10]

1) ***Presentation***, exchange of information about the system with presentations with stakeholders. There are 3 steps to presentation:
   a. **Describe ATAM itself,** The raters explain the method so that those who will be involved in the evaluation
   b. **Describe business drivers,** An appropriate system representative presents an overview of the system, its requirements, business objectives, and context, and the drivers of its architectural quality attributes
   c. ***Present the proposed architecture,*** The systems or software architect (or other key technical staff) presents the architecture
2) ***Investigation & analisys***, considering the need for key quality attributes and architectural approaches. In this group there are 3 steps:
   a. ***Identify architectural approaches,*** is an architectural approach identified by, but not analyzed.
   b. ***Generate a utility tree describing quality attributes,*** hierarchical list of quality requirements.
   c. ***Analyze architectural approaches,*** This results in differences in sensitivity points in which approaches affect attribute quality, which approach points affect some attributes, and the quality of the list of points that cause risk.
3) ***Testing***, check the results obtained with the needs of relevant stakeholders. In testing there are 2 steps, namely:
   a. ***Brainstorm and prioritize scenarios,*** Now that stakeholders are included, brainstorm both current use cases, expected future "change scenarios," and extreme "exploratory scenarios."
   b. ***Re-analyze architectural approaches and priorities,*** Now focus on the most important scenarios from step 7. If a scenario cannot be realized using the chosen architectural approach, this needs to be adjusted.
4) ***Reporting***, report the final results of the ATM to stakeholders. There is 1 step in the report group:
   a. ***Present consensus results,*** Based on the information collected at ATAM (style, scenario, attribute-specific

questions, utility tree, risk, location, cost).

3 RELATED WORKS

Hajar Hameed Adden in the year of 2019 [9] perform an analysis by testing one of the resistance patterns, namely the circuit breaker pattern and modifying the circuit breaker by adding the Markov Chain algorithm and given the name DFTM (Dynamic Fault Tolerance Model) with circuit breaker. The results of the analysis are stated that DFTM has better performance and reliability than the original circuit breaker, the study only tested the circuit breaker without testing other patterns such as timeout pattern, bulkhead pattern and retry pattern.

For research conducted Nabor C. Mendonca Carlos M. Aderaldo in the year of 2020 [1] conducted an analysis of resistance patterns namely circuit breakers and retry patterns using the checker model namely PRISM, the results of these stated that if both patterns were configured correctly it could reduce failures in communicating between clients and servers. In this study, testing by adding a circuit breaker and a retry pattern can reduce failures in communication between client and server compared to without using these two patterns and this study does not examine other patterns such as bulkhead patterns and timeout patterns.

Furthermore, for the research conducted by Fabrizio Montesi and Janine Weber in 2018 entitled from the decorator pattern to circuit breakers in microservices[11] conduct an analysis of the settings for the application of circuit breakers and implement it using the Jolie programming language. Then for the research conducted by Elena Troubitsyna in 2019, which proposed a systematic model for fault tolerance, in this study defines a generic modeling pattern that can be used in microservice driven models to assist in analyzing possible failures and improving QoS. The research did not explain how to test the circuit breaker and the results obtained after doing the test

Next is the research conducted by Kanglin Yin, Qingfeng Du in 2019 [8] namely defining microservice resilience refers to systematic studies in other scientific fields and this research proposes a Microservice Resilience Measurement Model to measure the resilience of a microservice. In this study, two patterns of resilience were used, namely the circuit breaker and the bulkhead pattern, but did not explain in detail the results after using the two patterns.

In previous studies, many have conducted analysis and testing of resistance patterns, for the pattern of resistance that has been tested the most is circuit breakers. In this study, we will analyze, and test resilience patterns also using the Architecture Tradeoff Analysis Method or commonly abbreviated as ATAM, for resilience patterns not only circuit breakers but with several other resilience patterns, namely bulkhead patterns, timeout patterns and retry patterns. As for the test experiment using concurrent user parameters, the lag time between users and the addition of response time on different microservices.

4 RESEARCH METHOD

4.1 Data Collection

Observations on this data collection are the data needed in making analysis and design at the company. The first step in collecting this data is by observing by observing the object of research and work processes in the company, namely observing the microservice architecture that is currently running.

Interviews were conducted in a structured manner with related parties, including the aircraft platform manager to find out the architecture that runs in the company. Furthermore, interviews were conducted with the flight booking team leader related to the problems faced when there was a communication failure between microservices.

4.2 Object of research

The research at this writing is a microservice that is located in an online travel agent company in Indonesia, the application is called a flight order manager where this microservice service has two important main functions as follows: placing an order with microservice insurance and making an order for aircraft reservations.

4.3 Design, Analysis and Evaluation using the ATAM method

In the process of designing, analyzing the architecture that runs on the company, and evaluating the pattern of resilience to overcome communication problems between microservices using the ATAM method.

This study, the design method uses several stages from ATAM, in this design uses the architectural proposal stages which produce images of the proposed architecture based on the 4 + 1 model or better known as the Architectural View Model.

For this research project, the author develops two microservices, namely microservice order manager and microservice insurance using the Java programming language assisted by the Spring Webflux framework to support reactive programming, for the database itself using mongodb. In the microservice order-manager, one layer or component is added before communicating between microservices, this layer uses a supporting library, namely resilience4j which aims to create a resilience pattern, namely circuit breaker, bulkhead pattern, timeout pattern and retry pattern, then microservice insurance is scenariod to fail system.

Furthermore, experiments and recordings will be carried out by gradually increasing the number of concurrent users starting from 100 users, 200 users, 300 users, 400 users, 500 users and 1000 users and then adding a response time delay of 500 milliseconds. Simulations with different number of concurrent users will be carried out with the help of JMeter7 tools as shown in Figure 1.

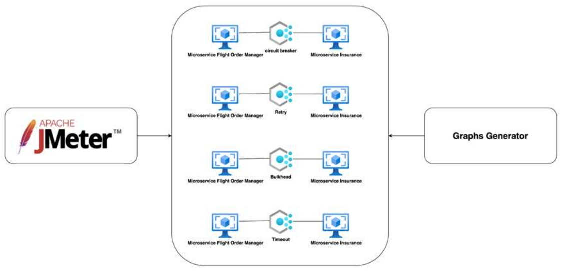

Figure 1 Simulation with Jmeter

With the Graph Generator, which is an additional plugin owned by the JMeter application, it can generate information in the form of csv and png, the information obtained includes the number of response times, error rates, throughput that can be used as parameters for the evaluation process.

In this study, the evaluation method used ATAM, the resilience patterns that were evaluated were Circuit Breaker Pattern, Bulkhead Pattern, Timeout Pattern and Retry Pattern using response time and throughput parameters, in this evaluation using three stages of the ATAM method, namely Brainstorm and Priority scenarios, Reanalyzing the approach and architectural priorities, and Present the consensus results.

5 METHODOLOGY THE ARCHITECTURE EVALUATION

In this section, we will discuss the design, analysis and evaluation carried out using the ATAM method as a method of developing the architecture. The stages of the ATAM method that will be carried out in this research start from the Presentation to the Reporting Stage

5.1 Step 1 Describe the ATAM method

At this stage, explaining to the company, to deal with problems related to communication failures between microservices, especially when the microservice flight order manager communicates with microservice insurance which often experiences high loads, the writing will test using the ATAM method.

5.2 Step 2 Describe Business Driver

At this stage, it explains what functional requirements are needed, and what test scenarios are needed to carry out testing in this research. The following are the results of the functional requirements and scenarios for testing.

At this stage there are three requirements needed, namely the system is still running or is experiencing a communication failure, whether there is a microservice that is off, the microservice is experiencing a high load. Then create microservices that can deal with communication failures and improve performance when they occur, and the last one implements the robustness pattern of Circuit Breaker Pattern, Bulkhead Pattern, Timeout Pattern, Retry Pattern which is expected to fulfill those needs.

5.3 Step 3 Present the proposed the architecture

this stage is to display the proposed architecture for the company, to display the architecture in this paper using a 4+1 architectural view, which will be displayed are Logical View, Process View, Development View, Physical View, and Scenarios.

In Figure 2 the following is a logical view using an activity diagram in the company. In the microservice order manager section, when the process of making aircraft orders from the process of creating order requests to the process of creating order data, and for the schedule selection process, the selection of aircraft has been carried out by other microservices. The author focuses on the process of making orders.

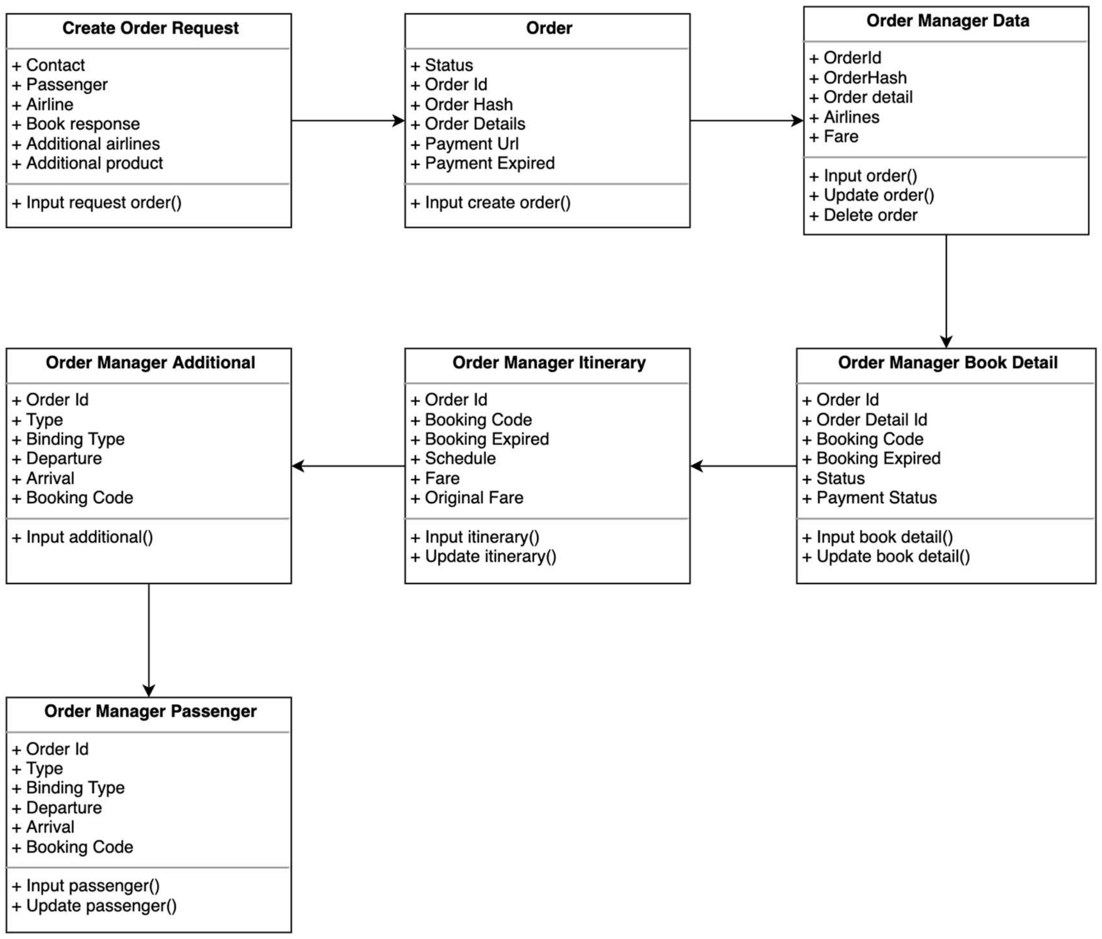

Figure 2 Class Diagram Proses Create Order

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

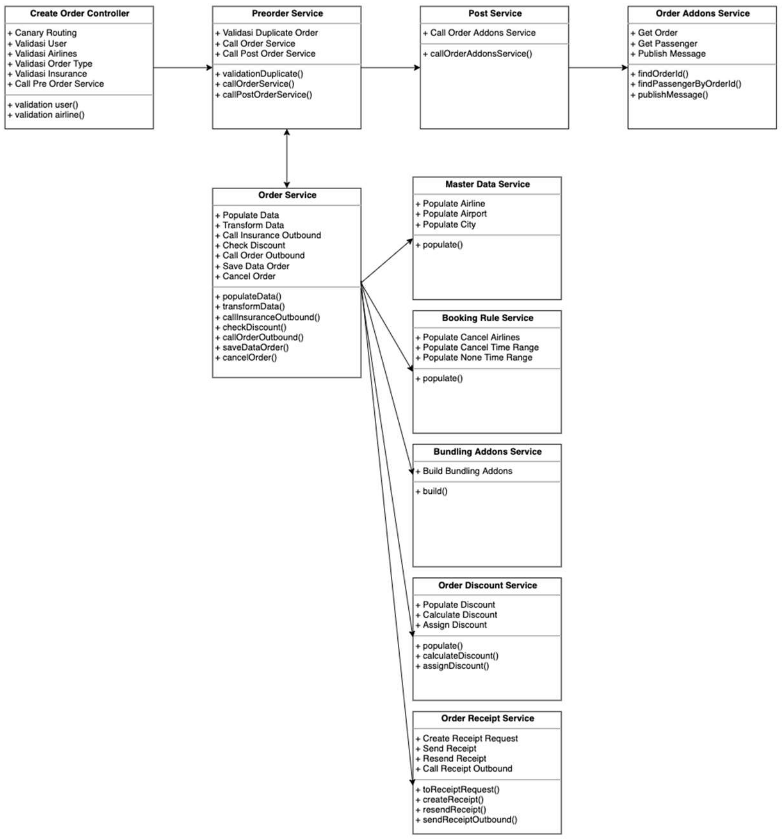

Figure 3 Class Diagram Controller dan Service Create Order

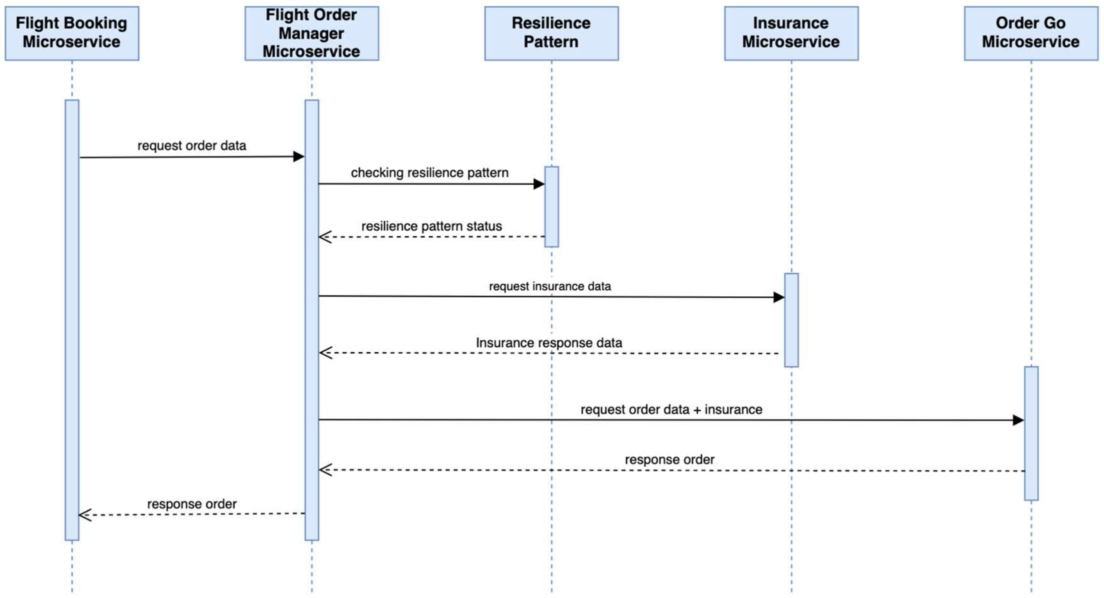

Figure 4 Sequence Diagram Create Order

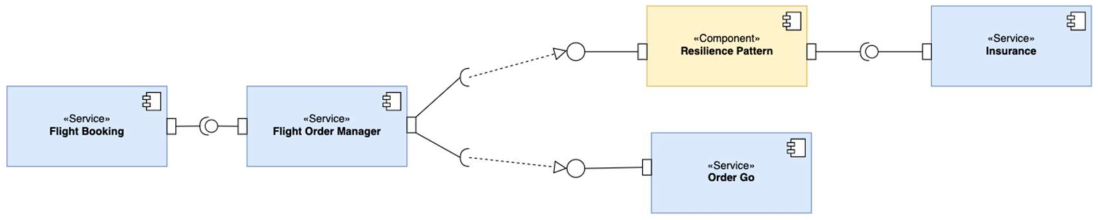

Figure 5 Component diagram relationship between microservices

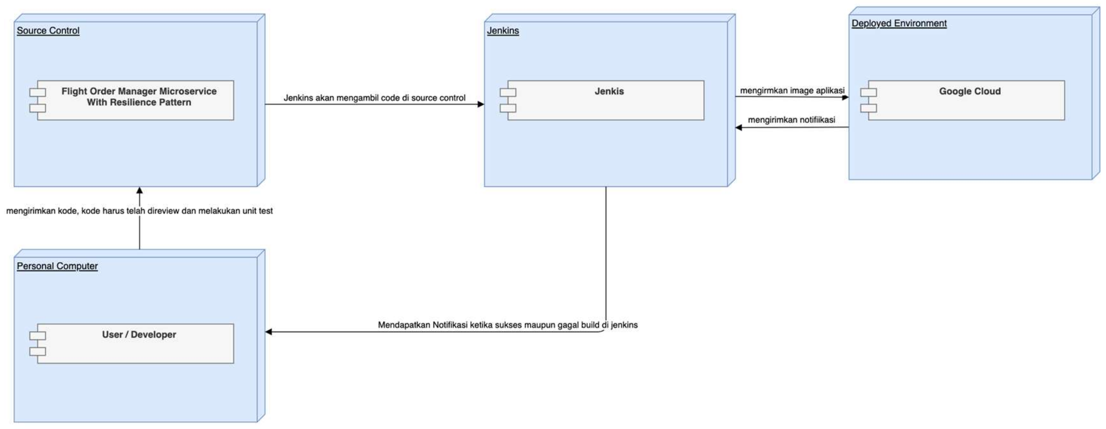
Source Control
Flight Order Manager Microservice With Resilience Pattern
Jenkins akan mengambil code di source control
Jenkins
Jenkis
mengirimkan image aplikasi
mengirimkan notifiikasi
Deployed Environment
Google Cloud
mengirimkan kode, kode harus telah direview dan melakukan unit test
Personal Computer
User / Developer
Mendapatkan Notifiikasi ketika sukses maupun gagal build di jenkins

Figure 6 Deployment Diagram

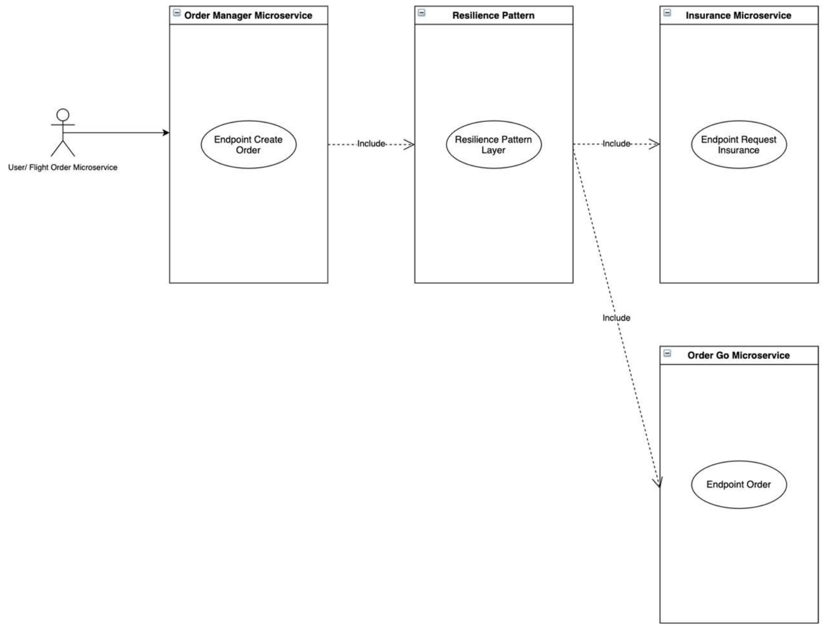

Figure 7 Use Case Microservice

In Figure 3 is a class diagram for the controller and service when making an order which consists of several services including order manager service, order service, post order service and so on.

In Figure 4 is a process view using a sequence diagram to explain how microservices communicate with each other when making orders, and when making insurance requests, until a new order id is formed for the transaction.

In Figure 5 is a development view that uses component diagrams to describe the components in the system along with the relationships and interactions that occur in microservices

In Figure 6 is a Physical view that uses a deployment diagram to describe the process of the relationship between software and hardware and each part of the device in a system. This system already uses continuous delivery, namely using the Jenkins application to help the process of deploying applications to the google cloud, because it uses the google cloud.

In Figure 7 is a scenario that uses a use case to describe the relationship process between microservices when making orders and insurance requests

5.4 Step 4 Identify Architectural Approaches

Based on the information obtained in the previous process, namely the proposed architectural process, in this process there is an additional layer resistance pattern on the microservice order manager before making a call to microservice insurance. To have a system that can be implemented and implemented, the system needs to be tested and based on the ATAM scenario that has been determined in the previous stage, namely Performance and Scalability.

The system needs to be tested to check whether they pass certain tests, the test is carried out using JMeter application, this test will display the Performance and Scalability of the system to ensure the performance is stable.

Table 1 General Scenario

QUALITY ATTRIBUTE	GENERAL SCENARIO	CONCRETE SCENARIO RECOMMENDATION	INITIAL DESIGN QUESTION
PERFORMANCE	service 1 makes an endpoint call to service 2, then receives less than n responses per millisecond	It may be necessary to specify a certain time limit	How resilience patterns can be improve performance?
			Is using a resistance pattern better than not using it?
SCALABILITY	Service will be called many requests n percent higher than usual, and service is still can work as expected	It may be necessary to specify a specific percentage of the increase request accepted	Does the resistance pattern need to be applied to increase availability?
	An example where a service would be inundated with high demand and requests simultaneously from time to time, and fixed service can work as expected	It may be necessary to specify the number of requests	Does the resistance pattern need to be applied to increase availability?

Table 2 Concrete Scenario Recommendation

QUALITY ATTRIBUTE	GENERAL SCENARIO	CONCRETE SCENARIO RECOMMENDATION
PERFORMANCE (P)	**P1 -** service 1 makes an endpoint call to service 2, then receive a response less than n per millisecond	Systems without resilience patterns and with resilience patterns Both will serve requests with less than 300ms response time when tested against 100 concurrent users.
SCALABILITY (SC)	**SC1** – An instance where the service will suddenly be flooded demand is n percent higher than usual, and service can still work as expected	Systems without resilience patterns and with resilience patterns Both will serve requests with a delay between users of 60 seconds when tested against 200 to 500 concurrent users.
	SC2 – An instance where the service will be flooded with a high amount requests simultaneously from time to time, and service can still work as expected	Systems without resilience patterns and with resilience patterns Both will serve requests with a delay between users of 2 seconds when tested against 200 to 500 concurrent users.
	SC3 – An example where the service will be incrementally called with increase simultaneously.	Systems without resilience patterns and with resilience patterns Both will serve requests with a delay between users of 2 seconds when tested against 1000 concurrent users and increased delay on the 2nd service by 500 milliseconds

5.5 Step 5 Generate Utility Tree

Based on the scenarios and architectural results, we will discuss Performance and Scalability, describing the Utility Tree to ensure each scenario in the Utility Tree has the appropriate stimulus and response.

With the proposed architecture, Prototype will be in ATAM approach. ATAM is a method for developing architectural designs. The focus of this evaluation will be placed on attributes, namely performance and scalability. The evaluation will be carried out in an evaluation based on a scenario where each Quality Attribute will have its own view. And then the researcher will provide the scenario template. The table below shows the general ATAM scenario that will be used to develop the prototype.

Table 2 states several scenarios that will be tried in this study. The interoperability quality attribute will focus on evaluating service-to-service communication in a microservice architecture. In addition, the quality of performance attributes will be discussed whether the service has a good performance. Then the scalability of the attribute will develop whether the microservice remains responsive when suddenly getting requests many times more than usual. Then this general scenario will be reduced to a concrete evaluation scenario for further evaluation. The Concrete Scenario Recommendation is derived from the General Scenario and is added based on the existing Quality Attributes.

Table 3 states the Concrete Scenario Recommendations which are extended from the General Scenario. Quality Attribute Scalability is defined to be able to maintain microservices when experiencing a growing number of requests to serve users given a sudden increase in demand without impacting the performance of microservices. Then Quality Attribute Performance is defined to keep the system able to serve music requests at any time. The system scenario will also be tested with several tests including trials to be given related to the different scalability and availability loads on the system. Load testing will be carried out using jmeter.

5.6 Step 6 Analyze Architectural Approaches

In the process of designing, analyzing the architecture that runs on the company, and evaluating the pattern of resilience to overcome communication problems between microservices using the ATAM method.

To have a successful and successful system, the system needs to be tested and tested. This architecture analysis is based on the ATAM scenario in the previous process. While the scenario is about performance and scalability, systems need to be tested to check whether they pass certain tests. The tests carried out are ramp ups and test tests. The two tests will be the performance and scalability of the system to ensure stable performance.

The first test carried out is a load test with certain concurrent users. To find out the average response time when tested against 100 concurrent users for 1 minute. The following is a table of results from these tests.

Based on table 3, what is done in the table shows that the pattern of resistance using a circuit breaker produces the smallest response, which is 266 milliseconds, while for throughput it does not produce a much larger value of 1.6 to 1.7 milliseconds, the following are the results of the analysis based on the quality of the Performance P1 attribute.

In table 4, the second test is carried out, namely with the same test, namely load testing with simultaneous user time, a pause between users of 60 seconds and the number of users from 200 to 500. the smallest response time is 192, 183, 209, 213 milliseconds and the next smallest response time is the resilience pattern with the timeout pattern producing 264, 272, 269, 296. Meanwhile, throughput does not produce much value.

Next, do the third test, which is with the same test, namely load testing with simultaneous user time, a pause between users of 2 seconds with the number of users from 100 to 500. The following are the results of the test

Based on the test in table 5, the circuit breaker produces the fastest response, and the largest throughput is 884, 2289, 3386, 4252 and 3449 while the throughput is 33.1, 43.4, 51.5, 55.9 and 74.0, for the fastest and the next fastest response is the timeout pattern. In this test the experimental pattern has the lowest and smallest values for both the response time and throughput and there is an error rate when the number of users is 500, namely the system without a pattern is 1% and the experimental pattern is 2%

Table 3 First Test

User	***Application layer***	***Average* (ms)**	***Throughput* (ms)**
100	Without Resilience Pattern	302	1,7
100	Resilience pattern - Circuit Breaker	266	1,7
100	Resilience pattern - Retry Pattern	2478	1,6
100	Resilience pattern - Bulkhead Pattern	341	1,7
100	Resilience pattern – Timeout Pattern	384	1,7

Table 4 Second Test

User	*Application layer*	*Average* (ms)	Min	Max	Error (%)	*Throughput* (ms)
200	Without Resilience Pattern	302	169	2331	0%	3.3
	Resilience pattern - Circuit Breaker	**195**	140	606	0%	3.3
	Resilience pattern - Retry Pattern	2497	2274	3018	0%	3.2
	Resilience pattern - Bulkhead Pattern	271	124	1097	0%	3.3
	Resilience pattern – Timeout Pattern	264	129	1378	0%	3.3
300	Without Resilience Pattern	277	176	1405	0%	5.0
	Resilience pattern - Circuit Breaker	**183**	127	548	0%	5.0
	Resilience pattern - Retry Pattern	2540	2264	3499	0%	4.8
	Resilience pattern - Bulkhead Pattern	266	143	754	0%	5.0
	Resilience pattern – Timeout Pattern	272	171	879	0%	5.0
400	Without Resilience Pattern	289	165	1623	0%	6.7
	Resilience pattern - Circuit Breaker	**209**	89	1340	0%	6.7
	Resilience pattern - Retry Pattern	2683	2349	3283	0%	6.4
	Resilience pattern - Bulkhead Pattern	292	171	856	0%	6.6
	Resilience pattern – Timeout Pattern	269	140	671	0%	6.7
500	Without Resilience Pattern	386	161	2245	0%	8.3
	Resilience pattern - Circuit Breaker	**213**	116	1067	0%	8.3
	Resilience pattern - Retry Pattern	2900	2309	3580	0%	8.0
	Resilience pattern - Bulkhead Pattern	340	169	1649	0%	8.3
	Resilience pattern – Timeout Pattern	296	123	768	0%	8.3

Table 5 Third Test

User	Application layer	Average (ms)	Min	Max	Error (%)	Throughput (ms)
100	Without Resilience Pattern	*2392*	*1391*	*3894*	*0%*	*21.4*
	Resilience pattern - Circuit Breaker	*884*	*309*	*1501*	*0%*	*33.1*
	Resilience pattern - Retry Pattern	*5270*	*3282*	*7315*	*0%*	*11.5*
	Resilience pattern - Bulkhead Pattern	*1258*	*492*	*2308*	*0%*	*26.2*
	Resilience pattern – Timeout Pattern	*2007*	*382*	*2949*	*0%*	*25.6*
200	*Without* Resilience Pattern	*2402*	*661*	*4076*	*0%*	*36.4*
	Resilience pattern - Circuit Breaker	*2289*	*494*	*3850*	*0%*	*43.4*
	Resilience pattern - Retry Pattern	*10368*	*5162*	*14068*	*0%*	*12.7*
	Resilience pattern - Bulkhead Pattern	*2822*	*1404*	*4665*	*0%*	*33.7*
	Resilience pattern – Timeout Pattern	*2367*	*992*	*3817*	*0%*	*43.3*
300	*Without* Resilience Pattern	*3783*	*1402*	*6920*	*0%*	*37.9*
	Resilience pattern - Circuit Breaker	*3386*	*916*	*5220*	*0%*	*51.5*
	Resilience pattern - Retry Pattern	*14963*	*5207*	*20630*	*0%*	*13.5*
	Resilience pattern - Bulkhead Pattern	*5820*	*315*	*7558*	*0%*	*34.5*
	Resilience pattern – Timeout Pattern	*5397*	*618*	*7525*	*0%*	*36.7*
400	*Without* Resilience Pattern	*5027*	*1066*	*8458*	*0%*	*43.1*
	Resilience pattern - Circuit Breaker	*4252*	*728*	*6730*	*0%*	*55.9*
	Resilience pattern - Retry Pattern	*20059*	*6942*	*27975*	*0%*	*13.6*
	Resilience pattern - Bulkhead Pattern	*8880*	*4146*	*11424*	*0%*	*34.8*
	Resilience pattern – Timeout Pattern	*6515*	*501*	*9549*	*0%*	*40.8*
500	*Without Resilience Pattern*	*5650*	*7*	*10742*	*2%*	*45.9*
	Resilience pattern - Circuit Breaker	*3449*	*45*	*6660*	*0%*	*74.0*

	Resilience pattern - Retry Pattern	*10279*	*334*	*22207*	*1%*	*21.2*
	Resilience pattern - Bulkhead Pattern	*5658*	*491*	*9084*	*0%*	*47.7*
	Resilience pattern – Timeout Pattern	*7102*	*155*	*11640*	*0%*	*41.6*

5.7 Step 7 Brainstorm and Prioritize Scenarios

At this writing add a new scenario, to ensure performance and scalability are appropriate, then perform the fourth test, namely with the same test, namely load testing with concurrent user time, a pause between users of 2 seconds with 1000 users and an additional 500 millisecond response time on the service.

The following are the results of testing the system without a pattern of resistance with 1000 users, the time lag between users is 2 seconds and the response time to the service is 500 milliseconds.

Based on system testing without a pattern of resilience, Figure 7 shows that the average response time is 13502 milliseconds, resulting in a throughput of 26.7 and in this test an error rate of 5%.

Based on testing the system with circuit breaker pattern resistance, Figure 8 shows that the average response time produced is 9253 milliseconds, resulting in a throughput of 50.6 and the test results in an error rate of 17%, the test results state that the system with circuit breakers has an average response time of - average compared to other tough patterns and the largest throughput.

Based on the system test with the resilience pattern retrieval pattern, Figure 9 shows that the average response time generated is 45980 milliseconds, resulting in a throughput of 7.7 and the test results in an error rate of 20%. From the test results, the retry pattern is the pattern with the longest average response time and the smallest throughput compared to other resilience patterns.

Based on testing the system with the resilience of the bulkhead pattern, Figure 10 shows that the average response time is 16574 milliseconds, resulting in a throughput of 24.8 and an error rate of 13%.

Based on testing the system with a resilience timeout pattern, Figure 11 shows that the average response time generated is 10676 milliseconds, resulting in a throughput of 47.9 and an error rate of 10%, the test results state that a system with a timeout pattern has an average reception time. second fastest and second largest throughput after circuit breakers.

The following are the results of the fourth test, in the table below it shows that the circuit breaker produces the fastest average response and the largest throughput compared to other resistance patterns. For the late response time and the smallest throughput is the retry pattern.

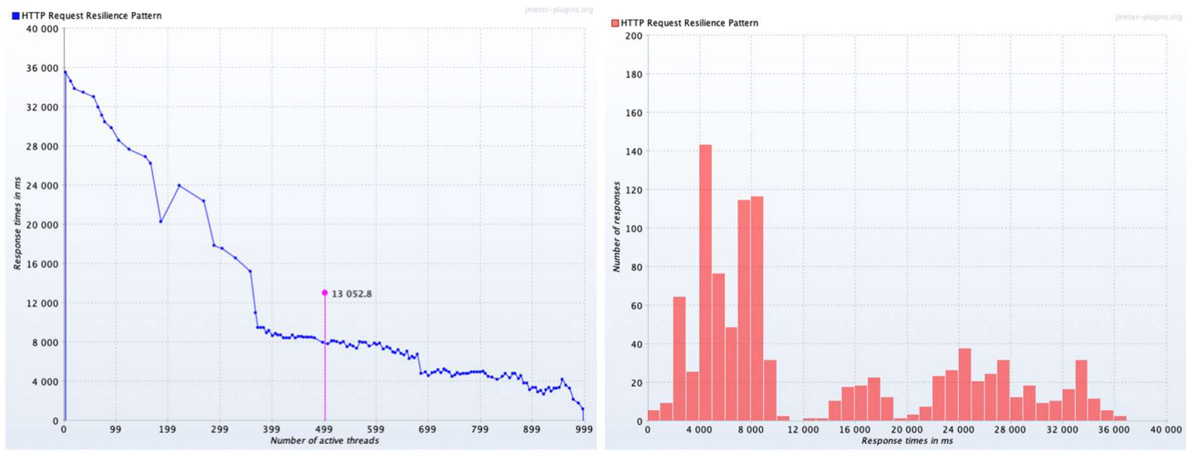

Figure 8 Without resilience pattern

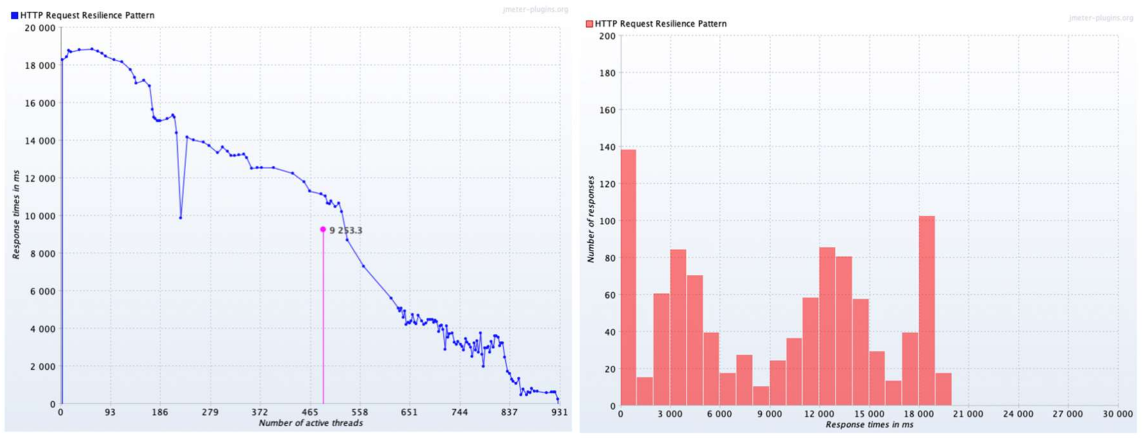

Figure 9 Circuit Breaker

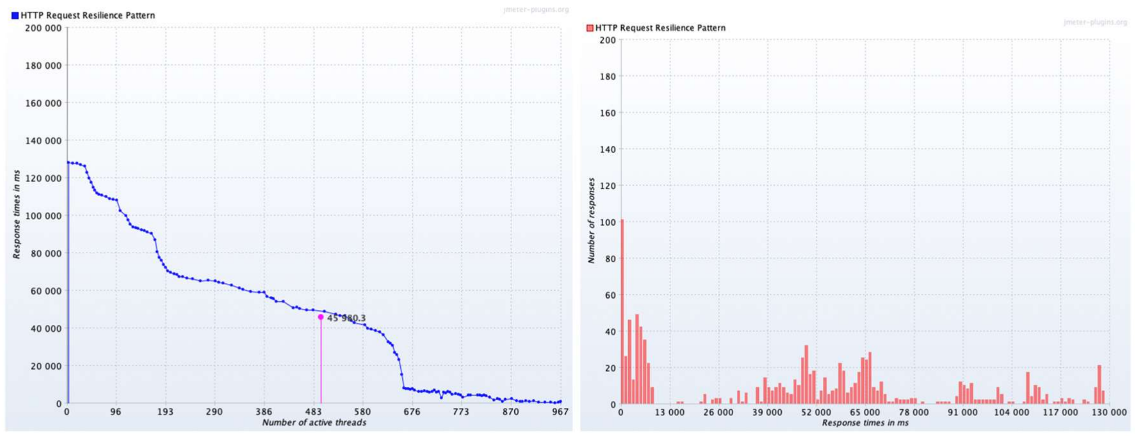

Figure 10 Retry Pattern

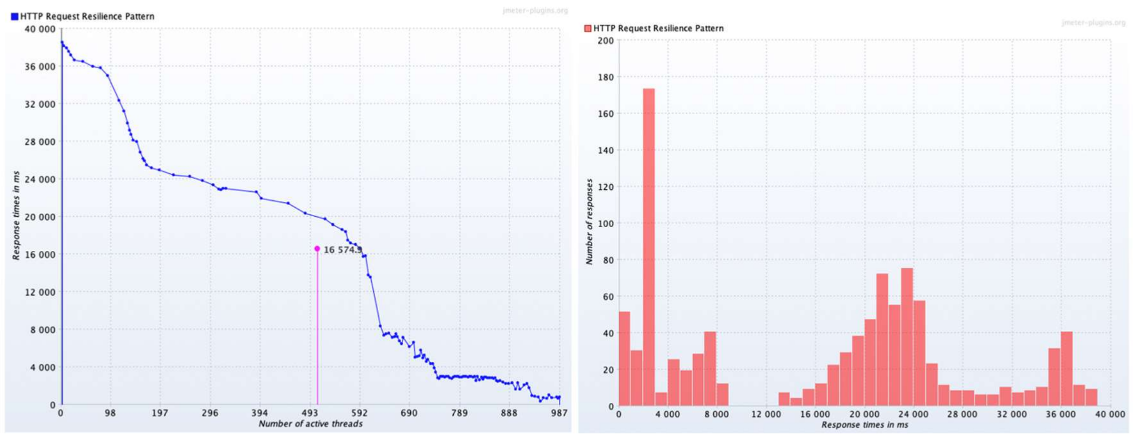

Figure 11 Bulkhead Pattern

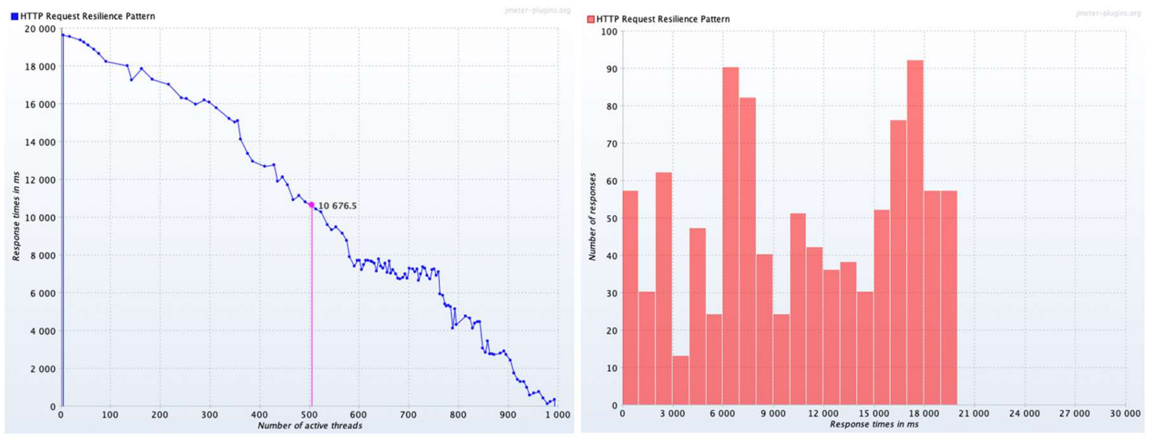

Figure 12 Timeout Pattern

5.8 Step 8 Re-analyze Architectural Approaches and Priorities

Based on the ATAM process that was carried out previously, it was found that with the addition of a resilience pattern it can provide a faster response time when a failure is found in communicating between microservices, but not all resilience patterns produce a faster response, there are even patterns of resilience that produce a faster response. slower than not using the resilience pattern, namely the retry pattern. This is very understandable because the concept of the retry pattern is when trying to repeatedly communicate to the microservice there is a failure to communicate. The fastest response time is circuit breaker and then followed by a timeout pattern.

5.9 Step 9 Present Consensus Results

This is the final step of the ATAM evaluation. Information collected during the evaluation. After going through several phases of ATAM, information is obtained that the performance and speed of responding to the microservice order manager when communicating with microservice insurance that is experiencing failure to provide a failed response by adding components or layer resistance patterns with the aim that the system is able to tolerate faults when errors occur. communicating between microservices obtained results which stated that with the addition of a layer resilience pattern could improve performance and response speed compared to without a layer resilience pattern.

6 DISCUSSION

An important part of building a resilient system, especially when functionality is spread across a number of different microservices [3] maybe there are several microservices that are experiencing up or down, this situation is more complex when the microservice already supports the reactive system because in the reactive system everything runs in asynchronous [4], this is the ability to degrade functionality in a more secure way. There are several patterns, which take together as measures for architectural security, that can be used to ensure that if something goes wrong, it doesn't cause lasting problems namely circuit breakers, bulkhead patterns, timeout patterns, and retry patterns. With this pattern, it is expected to solve the problem, namely a system that is able to tolerate faults [6]. This is in line with the manifesto reactive characteristics, namely resilience [5].

In previous research [9] [1] [11] [8] the purpose of the research is broadly in accordance with this research, namely to create a system that is able to survive in the face of a system failure, so that the system can run as expected.

Finally, based on the evaluation that was tested, the following comparisons were made between the author of the architecture and several other architectures shown in table 6

Table 6 Comparing with other Architecture

Work	Pattern			
	Circuit	Bulkhead	Timeout	Retry
[9]	yes	no	no	no
[1]	yes	no	no	yes
[11]	yes	no	no	no
[8]	yes	yes	no	no
this work	yes	yes	yes	yes

The main limitation of this research is that only four models are presented, namely circuit breaker, bulkhead pattern, timeout pattern and retry pattern without any modifications or only with basic usage. This research captures the response time and throughput of two microservices that are connected via the HTTP protocol and using REST, one of which is intentionally conditioned to experience a system failure.

For resilience patterns that have performance and response speed, the size of the test results that have been carried out for performance is based on the obtained throughput, if a large increase in throughput improves microservice performance, while for response speed based on the average response time obtained, the response will be smaller. time microservice. provide a faster response when communicating.

7 CONCLUSION

Based on the results of the analysis and analysis that the authors have described using the architectural tradeoff analysis method with the aim of comparing patterns of resilience that have good performance and fast responses, at the final stage of this paper the authors draw several conclusions according to the purpose of this paper. After being analyzed and explained in the previous process, the following conclusions are drawn:

Based on the previous ATAM process, it was found that with the addition of a resilience pattern it can provide a faster response time when a failure is found in communicating between microservices, but there are several patterns of resilience that produce a slower response compared to not using a resilience pattern, namely the retry pattern.

Based on the testing of the system with circuit breaker pattern resistance, the test results state that the system with circuit breaker has the fastest average response and the largest throughput compared to other tough patterns. Then the results of testing the system with the resilience timeout pattern show the test results stating that the system with the timeout pattern has the second fastest average response and the second largest throughput after circuit breakers.

In this paper, we only use simple usage in the application of resilience patterns. Feedback and ideas for the future. The author hopes that in the future we can add other pattern resilience models to be able to analyze, add or upgrade from existing pattern resilience models and combine several resilience patterns that are obtained according to existing business functions.

REFERENCES:

[1] N. Mendonca, C. Mendes Aderaldo, J. Camara, and D. Garlan, "Model-based analysis of microservice resiliency patterns," *Proc. - IEEE 17th Int. Conf. Softw. Archit. ICSA 2020*, no. February, pp. 114–124, 2020, doi: 10.1109/ICSA47634.2020.00019.

[2] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," 2016, doi: 10.1109/SOCA.2016.15.

[3] M. Fowler, "Microservice Trade-Offs," 2015. https://martinfowler.com/articles/microservice-trade-offs.html (accessed Apr. 25, 2021).

[4] M. Bernhardt, *Reactive Web Applications: Covers Play, Akka, and Reactive Streams*, 1st ed. Manning Publications, 2016.

[5] G. Jansen and P. Gollmar, *Reactive Systems Explained*, 1st ed. United States of America: O'Reilly Media, Inc, 2020.

[6] A. Sari and M. Akkaya, "Fault Tolerance Mechanisms in Distributed Systems," *Int. J. Commun. Netw. Syst. Sci.*, vol. 08, no. 12, pp. 471–482, 2015, doi: 10.4236/ijcns.2015.812042.

[7] E. Troubitsyna, "Model-Driven Engineering of Fault Tolerant Microservices," *Fourteenth Int. Conf. Internet Web Appl. Serv. (ICIW 2019)*, no. c, pp. 1–6, 2019, [Online]. Available: https://www.thinkmind.org/index.php?view=article&articleid=iciw_2019_1_10_20069.

[8] K. Yin, Q. Du, W. Wang, J. Qiu, and J. Xu, "On representing and eliciting resilience requirements of microservice architecture systems," *arXiv*, no. Ddd, pp. 1–15, 2019.

[9] H. Hameed Addeen, "A Dynamic Fault Tolerance Model for Microservices Architecture," South Dakota State University, 2019.

[10] D. Brahneborg and W. Afzal, "A Lightweight Architecture Analysis of a Monolithic Messaging Gateway," *Proc. - 2020 IEEE Int. Conf. Softw. Archit. Companion, ICSA-C 2020*, pp. 25–32, 2020, doi: 10.1109/ICSA-C50368.2020.00013.

[11] F. Montesi and J. Weber, "From the decorator pattern to circuit breakers in microservices," *Proc. ACM Symp. Appl. Comput.*, pp. 1733–1735, 2018, doi: 10.1145/3167132.3167427.