ISSN 2085-4552

Development of Restful API Mental Health Application with Microservices Architecture Using Google Cloud Platform

Muhamad Zidan[1], Is Mardianto[2], Agus Salim[3]
[1,2,3] Universitas Trisakti, Jakarta, Indonesia
[1] aldy10ball@gmail.com, [2] mardianto@trisakti.ac.id, [3] agus@trisakti.ac.id

Accepted 08 January 2025
Approved 24 June 2025

Abstract— **Circle application is an application on the Android operating system that is engaged in the field of mental health. The application requires a server that receives requests to fulfill the features contained in the application, therefore a server is needed one that can be scaled easily and quickly so that there are no long delays in requests, and it is hoped that the code base is easy to maintain and can be rapidly deployed.**

***Index Terms*— *Cloud Computing, Google Cloud Platform, Microservices, Docker*.**

I. Introduction

Mental health is a vital aspect of a person's life. Mental health is no less important than physical health for a person to have an everyday life and be able to adapt to the problems encountered throughout his life. According to WHO, mental health is a condition of well-being that individuals realize, in which they can manage reasonable life stress, work productively and productively, and participate in their community [1]. One in three teenagers (34.9%), equivalent to 15.5 million Indonesian teenagers, had one mental health problem in the last 12 months [2]. Of all primary caregivers who stated that their teenager needed help, more than two-fifths (43.8%) reported that they did not seek help because they preferred to handle the teenager's problems themselves or with support from family and friends. [2].

From the previously mentioned problems, there is the Circle application, Circle is a mental health application that has features, namely meditation, support groups, and online counseling with professional psychologists.

The application has a support group search feature, recommendation system, and payment system requires a server to process requests sent from the Circle application, so the application requires a scalable infrastructure, has an acceptable response time and is easy to maintain.

There are four main problems in developing and deploying a web application [3], namely: (1) Dependencies: a software depends on much other software, especially in the form of libraries for certain specific. (2) Incomplete documents or not solving initial installation and operational problems. (3) Code rot. Different versions of the library, operating system (kernel), or development language (interpreter) can also differ in the results provided by the application. For example, a bug update to a kernel or library can make software created to handle previous errors create new problems. (4) Barriers to adoption and reuse of pre-existing solutions.

The difficulties present in the above traditional approaches can be solved by virtualization technology. According to Zhang [4], virtualization is an integral part of modern cloud infrastructure such as Amazon's Elastic Compute Cloud (EC2) and Google's App Engine.

With the use of container-based virtualization technology in the form of Docker, web application development has several advantages [5], namely, making portable applications, more efficient use of computer resources, lightweight, fast, and suitable for developing microservices architecture.

This research improve reliability, fault tolerance, scalability and Flexibility with the use of virtualization technology and utilizing Google Cloud Platform robust infrastructure.

II. Literature Review

There are studies conducted by previous researchers who handled cases similar to this research. Here is a list of prior research used in this research as a reference in building solutions.

- Qalam AIlmiah dan Riko Virgiawan Z. PERANCANGAN ARSITEKTUR BACKEND MICROSERVICE PADA STARTUP CAMPAIGN.COM, The problem experienced by this research is that the system design built at campaign.com still uses a monolithic architecture. The user interface, logic processing, and data access are combined into one program and placed in one database. So it isn't easy to maintain. The advantages in this study, namely the provision of methods that are easy to understand and implement in different studies, while the weaknesses in this study are that the endpoint is only accessed by the web, and the docker configuration and implementation are still done manually, there is no stress testing.

- Sinambela A dan Farady Coastera F, IMPLEMENTASI ARSITEKTUR MICROSERVICES PADA RANCANG BANGUN APLIKASI MARKETPLACE BERBASIS WEB. The problem experienced by researchers is the use of monolith architecture which makes the research server consume significant computing resources and also causes obstacles in developing new features on the system. The advantage of this research is the creation of an API gateway that can unite separate services. In contrast, the disadvantages contained in this research are deployments that are still done manually and there is no stress testing.
- Jhay Shah dan Dushyant Dubaria. Building modern clouds: Using Docker, Kubernetes Google Cloud Platform. The problem experienced by researchers is that researchers want to find a faster way of deployment, and need a facility that can scale well. The advantages of this research discuss the use of Kubernetes that can manage many docker containers efficiently and are fully documented. In contrast, the disadvantages of this research are the lack of performance tests on the infrastructure that has been created.

A. *Microservice Architecture*

Unlike a monolithic application, Microservice means dividing an application into smaller, interconnected services. Each microservice is a small application with a hexagonal architecture consisting of logic and various adapters, as illustrated in Figure 1.

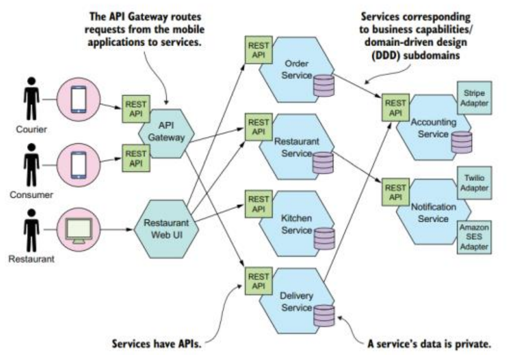

Figure 1. Microservices Architecture

Microservice architecture is a more scalable distributed alternative that provides more focused and specific services. Large problems will be broken down into several small solutions organized into a single service, where each service has its responsibilities. With this approach, an information system will consist of several services that can be managed and distributed independently, making it easier for the system to adapt to changing needs [6].

B. *RestFul API*

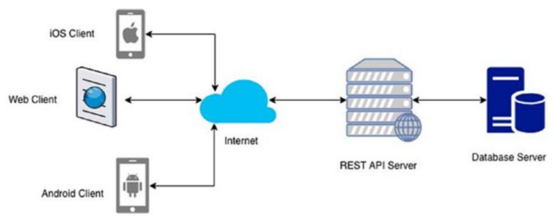

Figure 2. REST Architecture Diagram

Representational State Transfer (REST) is a style of software architecture for web services that provides standards for data communication between different types of systems [11]. In simple terms, REST is a standard for exchanging data over the Web for interoperability between computer systems. REST allows us to distinguish between client and server and implement client and server independently. The most important feature of REST is its statelessness, which means that neither client nor server needs to know each other's status to be able to communicate [7].

C. *Docker*

Docker is an open platform that can be used to build, distribute, and run applications and has a portable, lightweight packaging tool known as the Docker Engine. Docker also provides Docker Hub, a cloud service for sharing applications. The cost of using Docker containers is much more efficient than ordinary virtual machines. This reduces the cost of building applications on cloud computing platform providers [8]

Applications built on Docker are packed with all the dependencies they need into a standard form called a container. These containers continue to run in isolation on top of the operating system kernel, Docker containers can be easily deployed to cloud-based environments [3].

Containerization is a way of running multiple software applications on the same machine. Each is run in an isolated environment called a container. A container is a closed environment for software. It combines all the files and libraries the application needs to function correctly. Multiple containers can be deployed on the same machine and share resources. Docker uses images to create containers [9].

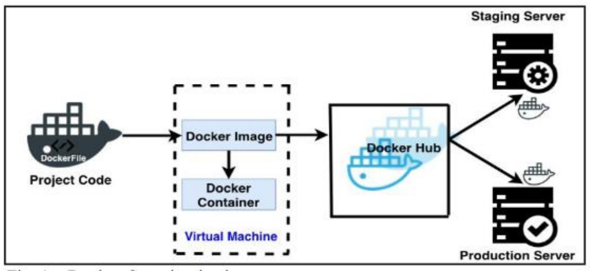

Figure 3. REST Docker Containerization

D. Google Cloud Platform

Google Cloud Platform (GCP) is a cloud computing service product owned by Google. GCP includes public cloud infrastructure and enterprise versions of G-Suite of Android, Chrome, and application programming interfaces for machine learning and enterprise mapping services. Google Cloud Platform provides over 100 services, such as computing, Networking, Storage and Databases, and others [10].

E. Cloud Build

Cloud Build is a service that runs app builds or dockers on Google Cloud. Cloud Build can import code from various repositories or cloud storage spaces, resulting in predefined specifications, and generate artifacts such as Docker containers or Java archives [11]. Cloud Build can easily be integrated into various public or private repositories such as existing repositories on GitHub this integration is done by Cloud Build utilizing pub-sub actions contained in the repository, Cloud Build can also automatically perform unit tests and integration tests to other Google Cloud Platform services and also Cloud Build has an auto deployment feature where the results of the build can be directly deployed to several Google Cloud services Platforms like Cloud Run.

F. Cloud Run

Cloud Run is a Serverless computing platform entirely managed by Google, Cloud Run can help run applications in highly scalable containers and can be called via web requests or from webhooks. Built on Knative, it enables high application portability. With Cloud Run, users can automatically scale up or down from zero to N. Cloud Run services are regional and automatically replicated across multiple zones. Cloud Run provides out-of-the-box integrations with Cloud Monitoring, Cloud Logging, Cloud Trace, and Error Reporting to monitor application performance [12].

Cloud Run offers some configurations that can be changed in each of the revisions of the container that we deploy, some of the configurations namely: (1) Port which request will be sent to the port of the container, (2) CPU allocation, which we can toggle if CPU is only allocated during request processing or CPU is always allocated, (3) Capacity of the Container including Memory, numbers of vCPUs, how long the request should be timed out, and maximum number of concurrent requests per container instance, (4) Execution environment which is First generation who prioritize speed from a cold start and Second generation that can utilize files system, full Linux compatibility, faster CPU, and Network performance, (5) Autoscaling that Bounds the number of container instances for the created container revision, and Lastly (6) Environment Variables

G. Cloud Storage

Cloud storage is one of Google Cloud Platform services to store unstructured data, Cloud storage uses an object-based storage system. An object is an immutable data consisting of files of any format. You store objects in containers called buckets. All buckets are associated with a project, and you can group your projects under an organization. Every project, bucket, and object in Google Cloud is a resource Google Cloud, as are things like Compute Engine instances. Cloud storage has several classes for its storage types:

Table 1. Cloud Storage Classes

Storage Class	**API Name and CLI**	**Minimum Duration Storage**	**Typical Monthly Availability**
Standard Storage	STANDARD	None	>99.99% di multi-regions dan dual-regions, 99.99% di regions
Nearline Storage	NEARLINE	30 Days	99.95% di multi-regions dan dual-regions, 99.9% di regions
Coldline Storage	COLDLINE	90 Days	99.95% di multi-regions dan dual-regions, 99.9% di regions
Archive Storage	ARCHIVE	365 Days	99.95% di multi-regions dan dual-regions, 99.9% di regions

H. Flask

Flask is a lightweight microframework for web applications built on Python, which provides an efficient framework for building web-based applications that use the flexibility of Python and strong community support with scaling capabilities to serve millions of users [7]. Flask has two main components, Werkzeug and Jinja. While Werkzeug is responsible for providing routing, debugging, and Web Server Gateway Interface (WSGI). Flask leverages Jinja2 as a template engine. Natively, flask does not support database access, user authentication, or other high-level utilities. Still, it provides support for the integration of extensions to add all such functionality, making Flask a micro yet production-ready framework for developing applications and web services. A simple flask of an application can fit into a single Python file or it can be modulated to make the application production-ready. The idea behind Flask is to build a good foundation for all applications leaving everything else on the extension [7].

I. Firebase

Firebase is a BaaS (Backend as a Service) service provided by Google, Firebase is considered a web application platform. Firebase helps developers build quality apps quickly. then Firebase stores the data in JavaScript Object Notation Format (JSON) which does not use queries to insert, update, delete, or append data to it. The system's backend is used as a database to store data [13].

J. Firebase Auth

Firebase Auth supports social sign-in features such as Facebook, Google, GitHub, and Twitter. It is a service that can authenticate users by using client-side code and is a paid service. Firebase Auth also includes a user management system where developers can enable user authentication with email and password logins stored with Firebase [13].

K. Firestore

Firestore is a flexible and scalable mobile, web, and server development database from Firebase and Google Cloud Platform. Like Firebase's real-time databases, Firestore syncs your data across all client apps via real-time handlers and provides offline support for mobile and web. This way, you can build responsive and efficient applications without network latency or internet connection. Firestore also offers seamless integration with other Firebase and Google Cloud products, including Cloud Functions [14]. Firestore is a cloud-hosted NoSQL database that can be accessed directly by your Apple, Android, and web apps through native SDKs. Firestore is also available in the native SDKs of Node.js, Java, Python, Unity, C++, and Go, in addition to the REST API and RPC API [14].

III. METHODOLOGY

The steps in this research are carried out in the following stages:

1. Conduct interviews and surveys to the target market to get the wants and needs of the target market. This data retrieval is carried out intermittently to get an idea of what features need to be made.
2. Based on the results of interviews and surveys on the target market and get the features that are wanted, researchers design UML for each use case and dissect it into microservices respectively.
3. After determining the microservice division, researchers design the use of Google Cloud Platform services which are made into a diagram, the services used by researchers like Cloud Build, Cloud Storage, Cloud Run, Firebase, and Firestore.
4. Build API endpoints based on predefined use cases and microservices, at this stage, researchers build a structure and codebase in a local environment.
5. Testing the code base in the local environment to ensure the features in the microservice section have run well under the system design that has been made, with the black-box testing method.
6. Deploy code in the local environment to the Google Cloud Platform infrastructure with containers. At this stage, researchers directly use the Google Cloud Platform infrastructure to deploy code in the GitHub repository and install the container in the cloud run.

Load test API endpoints that have been deployed to the Google Cloud Platform. This stage is necessary to simulate real traffic and ensure that the system is not damaged under the specified load

IV. RESULT AND DISCUSSION

A. User Requirement

Based on the results of focus group discussions with several psychologists related to the use of support group features and mental health application features in general by users in building systems in the Circle application, here are some user needs needed in the system in the Circle application:

1. Users can search for the desired support group and enter the group chat
2. Users can order psychologist services in the form of consultations and pay according to the services that have been chosen
3. Users can do support group activities in the available group chat.
4. User can register and log in to the application.
5. User can do screening test.
6. Caregivers can access the results of the user screening test.
7. Caregivers can create support groups and moderate chat rooms.

B. *System Requirement*

Based on the results of focus group discussions with several psychologists related to the use of support group features and mental health application features in general by users in building systems in the Circle application, here are some of the system that needs to be needed in the Circle application:

1. Restful API will be deployed on the Google Cloud Platform.
2. Restful API has a simple security system in the form of an API Key.
3. Restful API can be accessed via HTTPS request.
4. The features required in creating a Restful API are as follows:
   - Create a Support Group in the form of a chat room.
   - Search Support Group for Users.
   - Create a login token for the GetStream service.
   - Booking Psychologist services.
   - User payment handling.
   - Create and evaluate screening test results for Users.

For the specification of each existing docker container unit to create a restful API, a minimum of the following specifications are required:

1. 1 Core CPU
2. 512 MB RAM
3. Access Port 8080

Then there is also a software environment that must be prepared, namely:

1. Python 3.9
2. Python libraries: Tensorflow, Sastrawi, GetStream-Client, Midtrans, Flask, and Firebase.

C. *Use Case Diagram*

Use case diagrams to illustrate a graphical display of the functioning of a system. For an explanation of the role of each actor performed on the Circle application can be seen in the following image:

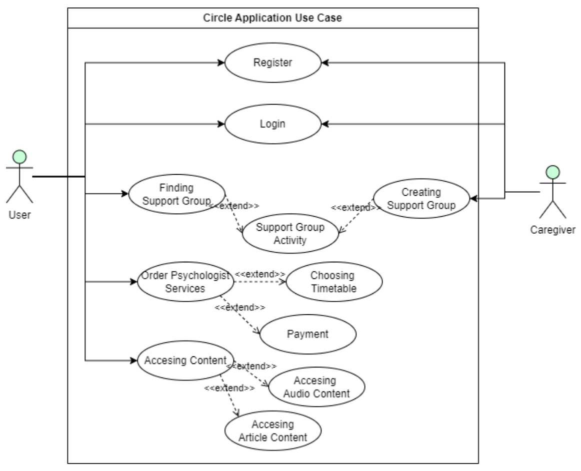

Figure 4. Use Case Diagram

D. *Activity Diagram*

The following stages of explanation of the role activities of each actor are contained in Figure 3.D.2 use case diagram:

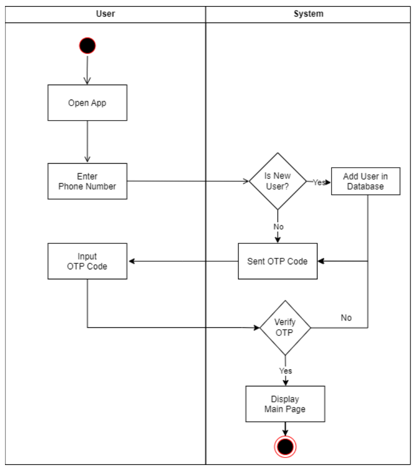

Figure 5. Activity Diagram: Log in and Register

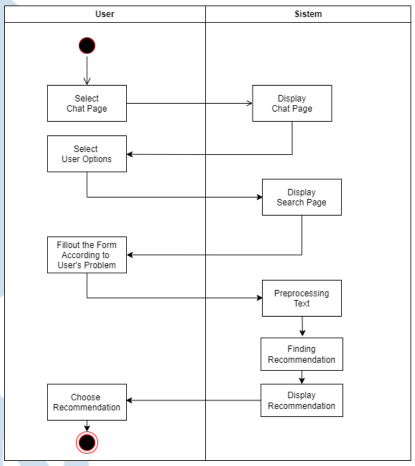

Figure 6. Activity Diagram: Search Support Group

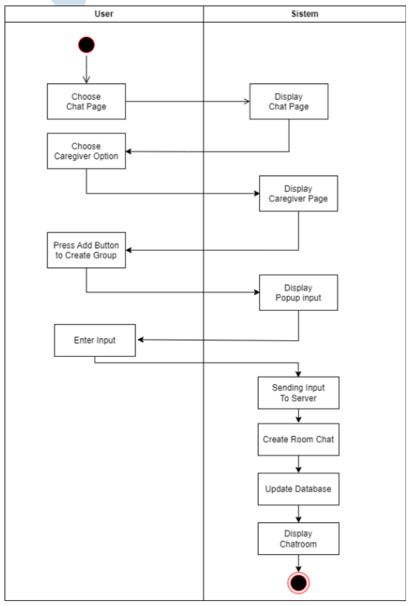

Figure 7. Activity Diagram: Support Group Creation

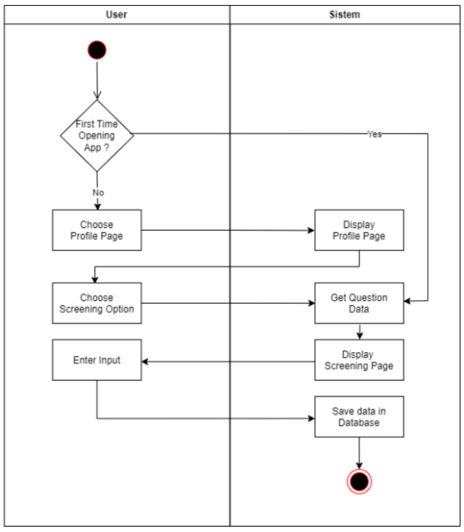

Figure 8. Screening: Psychological Tests

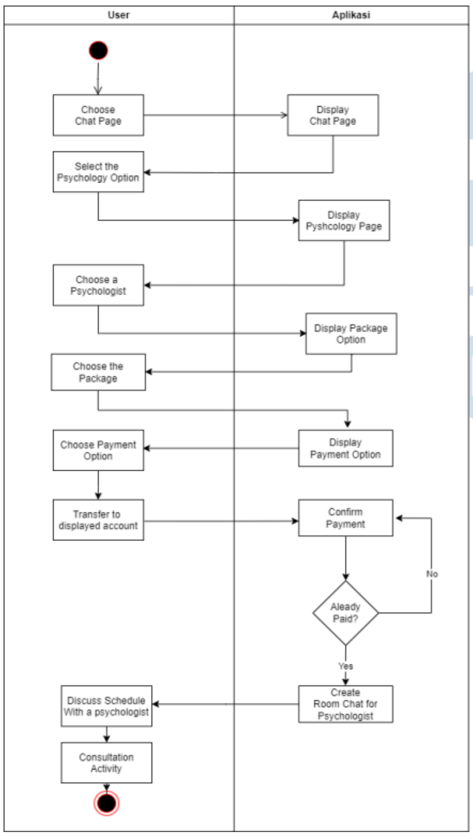

Figure 9. Psychologist Consultation

E. Microservices Architecture on Google Cloud Platform

Referring to user requirements and system requirements services in the Circle application can be divided into 4 systems, namely:

1. Chat Service
2. Payment Services
3. Search Service
4. Screening Test Service

With these 4 services can be made a picture of the system architecture in making it into Google Cloud Platform, here is an overview of the use of Google Cloud Platform products and microservices architecture that will be used:

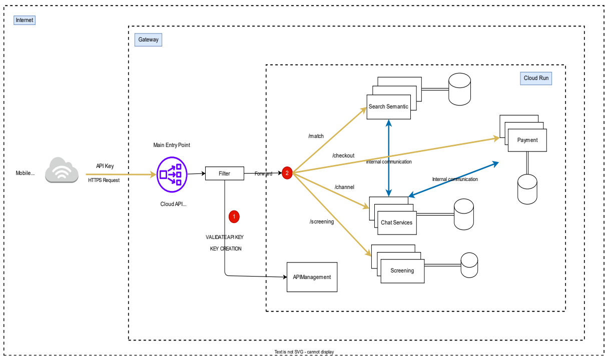

Figure 10. Microservice Architecture

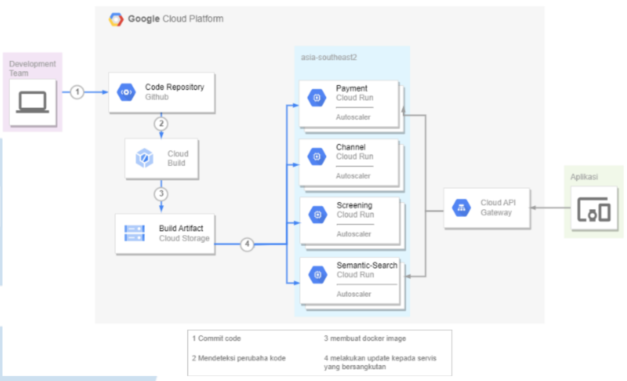

Figure 11. GCP Usage

In the GCP usage image above, there are several services that are used for Circle application needs, namely:

1. Cloud Build

   Cloud Build is used to create a Docker image that matches the contents of the repository on Github, each service has its repository so that they are mutually independent from other services

2. Cloud Storage

   Cloud Storage is used to store every Docker image version that Cloud Build has created, so this can help in storing stable backup versions

3. Cloud Run

   Cloud Run is used as a serverless platform which will be the main server for each of the microservices mentioned earlier, Cloud Run also has an autoscaler system so that it can multiply Containers so that it automatically scales requests received from the Circle Application, this happens on-demand so if there are no requests from the application, the resources from Cloud Run will not be used

4. Cloud API Gateway

   Cloud API Gateway allows us to manage URLs from different Cloud Runs into one URL domain making it easier for developers to access APIs, Cloud API Gateway also provides monitoring, alerts, logging, and tracing for each use

F. *Cloud Run Configuration*

Based on the Microservice Architecture in the previous figure we can utilize Cloud Run to divide the Services and use Firebase as the Database for each service, for the specification of the container for each service in Cloud Run is shown in the next tables

Table 2. Search Services Cloud Run Configuration

Search Service	
CPU allocation	CPU is only allocated during request processing
Startup CPU boost	Enabled
Maximum Concurrency per instance	80
Request timeout	300 seconds
Execution environment	Second generation
CPU	1
Memory	2 GiB
Minimum number of instances	0
Maximum number of instances	10
Port	8080
Build	Cloud Build
Environment Variables	Firebase_Key

Table 3. Chat Services Cloud Run Configuration

Chat Service	
CPU allocation	CPU is only allocated during request processing
Startup CPU boost	Enabled
Maximum Concurrency per instance	80
Request timeout	300 seconds
Execution environment	Second generation
CPU	2
Memory	512 MiB
Minimum number of instances	0
Maximum number of instances	5
Port	8080
Build	Cloud Build
Environment Variables	Firebase_Key, GetStream_Key

Table 4. Payment Service Cloud Run Configuration

Payment Service	
CPU allocation	CPU is only allocated during request processing
Startup CPU boost	Enabled
Maximum Concurrency per instance	80
Request timeout	300 seconds
Execution environment	First generation
CPU	1
Memory	512 MiB
Minimum number of instances	0
Maximum number of instances	5
Port	8080
Build	Cloud Build
Environment Variables	Firebase_Key, Midtrans_Key

Table 5. Payment Service Cloud Run Configuration

Screening Test Service	
CPU allocation	CPU is only allocated during request processing
Startup CPU boost	Enabled
Maximum Concurrency per instance	80
Request timeout	300 seconds
Execution environment	First generation
CPU	1
Memory	512 MiB
Minimum number of instances	0
Maximum number of instances	30
Port	8080
Build	Cloud Build
Environment Variables	Firebase_Key

From the tables above, we allocated the CPU and Memory of the container based on how expensive the computational power of each service is, from the tables we can conclude that the Search Service is the most demanding in terms of Memory which is caused by the Tensorflow Library.

With Cloud Run when there's a massive spike of request to the Search Service we can individually scale the necessary Service without affecting other services.

To save cost, all of the services have a minimum number of container instances set to zero which enables a cold start, the number of instances will go up if and only if there's a request coming to the port of the container by utilizing the on-demand feature of the container we can reduce the billing of the cloud significantly.

G. *Containerization*

Containerization is a technology that allows you to package an application and all its dependencies, libraries, and configurations into a single, isolated unit called a "container." This container can then be easily deployed and run consistently on any server that supports containerization without worrying about compatibility issues.

To create a container we need a configuration setup called Dockerfile, This file contains instructions on how to build your container step-by-step. It specifies the base image (a minimal operating system with pre-installed tools), adds your application code, and sets up the required configurations. For Circle Services we use the following setup:

```
From python:3.9.11

COPY src/ src/
COPY requirements.txt.

RUN pip install --upgrade cython
RUN pip install --upgrade pip
RUN pip install --no-cache-dir -r requirements.txt


WORKDIR /src

ENV PORT 8080

CMD exec gunicorn --bind :$PORT --workers 1 --
threads 8 app:app
```

From the Dockerfiles above we can configure the directory of the files by first copying the app files to the Docker directory, and then we can set up the environment suited to the requirements we need

H. Restful API Design

Based on activity diagrams, use case diagrams and microservices architecture design, several endpoints can be made that will be used in Circle Applications, namely:

Table 6. Restful API

No	Fungsi	URL	Parameter	Method	Expectation
1	Login and Register	/token	User id, password	GET	Return status and Token
2	Getting Screening Data	/screening	User_id,screening_name	GET	Return data about screening
3	Post the Result of Screening	/screening	User_id, screening_name, jawaban	POST	Return Feature Recommendation
4	Search Chatroom	/match	User_id, k_value, text	GET	Return Chatroom recommendation
5	Create Chat Room	/channel	User_id, Text, Title, Max_user	POST	Return Channel information
6	checkout	/checkout	Appointment_id, user_id	GET	Return URL and transaction token
7	Notification Handling	/notification	Transaction_id,Transaction_status, Order_id	POST	Change the order status and return status code

I. Blackbox Test

To test the various feature of the API that have been designed we need some tests to confirm it. Blackbox test is designed to test the whole infrastructure to check if the integration between the services is correct:

Table 7. Blackbox Test

URLs	Test Case	Expected Result	Tested results
/token	Correct *User ID* and *Password*	Return Token Session	Valid
	Inccorect *User ID* or Password	Return incorrect message	Valid
	Missing parameter	Return incomplete parameter Message	Valid
/screening (GET)	Correct *User ID* and *Screening Name*	Return Screening Question Details	Valid
	Incorrect *User ID* and *Screening Name*	Return Incorrect message	Valid
	Missing Parameter	Return incomplete parameter Message	Valid
/screening (POST)	Correct *User ID, Answer Screening Name*	Return Feature Recommendation	Valid
	Incorrect *User ID, Answer and Screening Name*	Return Incorrect message	Valid
	Missing	Return	Valid

	Parameter	incomplete parameter Message	
/match	Correct *user id, text* and *k_value*	Return list of recommended room id	Valid
	K_value is not in parameter	Return top 3 in the list of recommended room id	Valid
	Inccorect *User ID or K_Value*	Return Incorrect message	Valid
/Channel	Correct *User ID,Text*, *Title,* and Max_user	Return Channel Information	Valid
	Inccorect *User ID*	Return Incorrect message	Valid
	Missing Parameter	Return incomplete parameter Message	Valid
/checkout	Correct *User ID* an *Appointment ID*	Return Redirect URL and transaction token	Valid
	Inccorect *User ID* and *Appointment ID*	Return Incorrect message	Valid
	Missing Parameter	Return incomplete parameter Message	Valid
/Notification	Correct *Transaction ID* and *Transaction Status*	Return Received Status Message	Valid
	Incorrect *Transaction ID* and *Transaction Status*	Return Incorrect Message	Valid
	Missing Parameter		

J. Load Test

For the Load Test, we use Locust as the Python framework of choice, the infrastructure of the microservices will be tested again with 5000 virtual users in a span of 10 minutes, and the following figures will describe how many users will be connected to the server and how many requests have been requested:

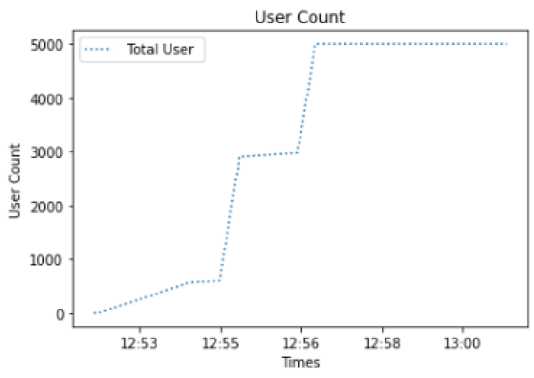

Figure 12. Total User Graph

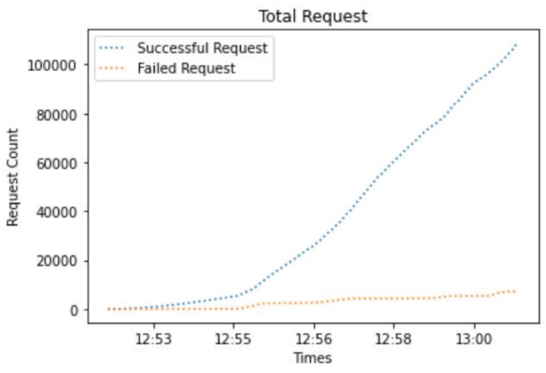

Figure 13. Total Request Graph

From the figures above, there are 5000 virtual users, which gradually increased from 0 to 5000 within 10 minutes. There’s a total of 115910 requests that are coming from the virtual users, as you can see from figure 4.J.2 There is some error that occurs when the user spike to more than 1000 and the total of the error is 7284 error, the detail of the error will be explained in the following table:

Table 8. Total Error Table

URL	Error	Number of Occurences
/match	HTTPError('429 Client Error: Too Many Request)	2820
/token	HTTPError('500 Server Error: Internal Server Error)	3957
/channel	HTTPError('429 Client Error: Too Many Request)	507

From the error table above, we can infer that there are 2 types of errors that happen during the load test, which is HTTP Error 429 and HTTP Error 500, the reason that HTTP Error 429 appear in /match and /channel is because of too many requests have been created for their endpoint, and the initial container can't scale fast enough to keep up with the demands. As for

HTTP error 500 is caused by the limit rate from the GetStream library that is used for getting the user a token to chat with other users, the limit rate is caused by the free-tier plan from the GetStream services.

The load test will also measure the response time of the entire service and each endpoint, for the parameter of the response time, we will measure the average response time and median response time, for the unit of the response time, we will use milliseconds, for the detail of the response time of the server will be illustrated in the following figures:

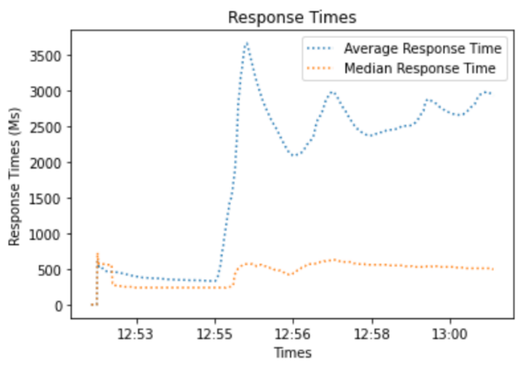

Figure 14. Response Time Graph

From the response time figures above, we can conclude that there's a huge spike when the use count reaches 5000, but it gradually lower, this is the result of the Cloud Run Autoscaler that automatically deploy new containers to serve the new Demands. The average response time when the server stabilize in 5000 users is 2817 milliseconds, we can dissect this further by exploring each endpoint response time as the following tables:

Table 9. Token URL Response Time

URL	/Token
Method	GET
Request Count	4999
Median Response Time	10000 ms
Average Response Time	10855 ms
Min Response Time	532 ms
Max Response Time	30011 ms
Average Content Size (Byte)	241 byte

Table 10. GET Screening URL Response Time

URL	/Screening
Method	GET
Request Count	26200
Median Response Time	260 ms
Average Response Time	384 ms
Min Response Time	219 ms
Max Response Time	5460 ms
Average Content Size (Byte)	4540 byte

Table 11. POST Screening URL Response Time

URL	/Screening
Method	POST
Request Count	25878
Median Response Time	280 ms
Average Response Time	400 ms
Min Response Time	223 ms
Max Response Time	5574 ms
Average Content Size (Byte)	16 byte

Table 12. Match URL Response Time

URL	/Match
Method	GET
Request Count	42012
Median Response Time	2700 ms
Average Response Time	4598 ms
Min Response Time	187 ms
Max Response Time	28278 ms
Average Content Size (Byte)	224 byte

Table 13. Channel URL Response Time

URL	/Channel
Method	POST
Request Count	8228
Median Response Time	5200 ms
Average Response Time	6731 ms
Min Response Time	185 ms
Max Response Time	30011 ms
Average Content Size (Byte)	547 byte

Table 14. Checkout URL Response Time

URL	/Checkout
Method	GET
Request Count	8526
Median Response Time	270 ms
Average Response Time	368 ms
Min Response Time	235 ms
Max Response Time	5597 ms
Average Content Size (Byte)	156 byte

Table 15. Notification URL Response Time

URL	/Notification
Method	POST
Request Count	67
Median Response Time	1500 ms
Average Response Time	2333 ms
Min Response Time	585 ms
Max Response Time	10044 ms
Average Content Size (Byte)	877 byte

From the seven tables, we can see a drastic change in the minimal response time to the max response time, this is because the warm up session with only 100 virtual user and gradually increase it to 5000 virtual user, we can use the minimal response time as the baseline and compare it to the median response time to get a sense how certain URL behave in this Load Test, Meanwhile the maximum response time corresponds to time when the user count spiked to 5000, and the server is rolling a new container, the time between finishing rolling the new container and the start of the request makes the max response time so much high compared to the average response time. In the previous load test we can see /match, /token, and /channel URLs have massive differences in minimal response time to average response time, for /match and /channel the high response time is caused by the resource-intensive system created by Tensorflow library and to prevent using too many resources of Cloud Run, limiting the number of containers is vital, as for /token case many of the error is caused by internal server error is caused by the limit rate from the GetStream library that is used for getting the user a token to chat with other users, the limit rate is caused by the free-tier plan from the GetStream services.

We can see how the container of each services automatically scale when facing a large request in the following figures:

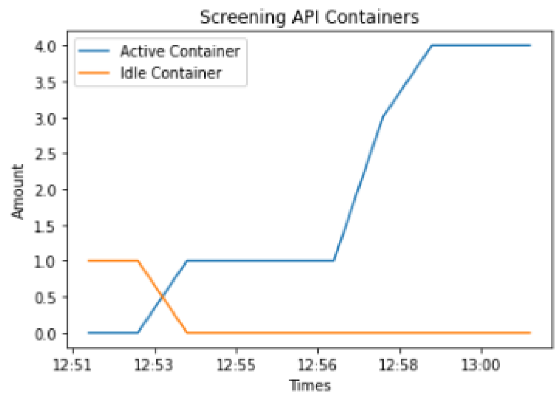

Figure 15. Screening Services Container Graph

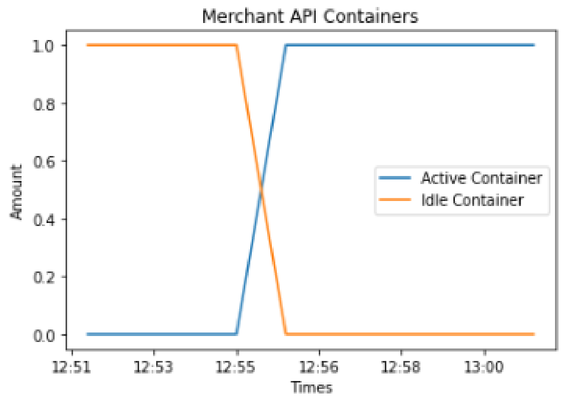

Figure 16. Merchant Services Container Graph

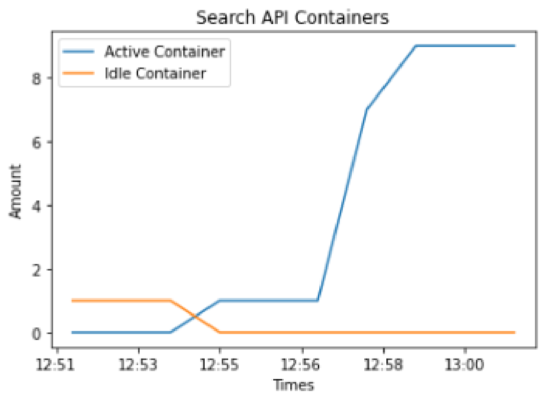

Figure 17. Search Services Container Graph

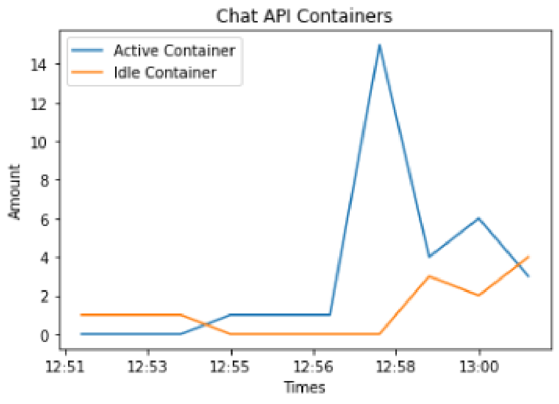

Figure 18. Chat Services Container Graph

From the figures above we can see how the Container automatically increases when the demand is high.

V. CONCLUSIONS

Based on the result of the research "Development of Restful API Mental Health Application with Microservices Architecture Using Google Cloud Platform" that has been carried out, there are several conclusions:

1. The making of microservices pattern in API for Circle Application that utilizes Google Cloud Platform as its cloud provider can be used and scaled based on the demand of the request from the application
2. Using a microservices pattern can remove dependencies from other services and as a result other services' load won't cause any disturbance to unrelated services, this also makes the scaling adjustable to only the most used service which raises the efficiency of the server and lowers the cost of the cloud computing cost
3. Using Cloud Build makes it easy to deploy a docker container and make changes to an existing container
4. Dockerization makes it easy to isolate bugs and faults within the system, and because we can set up the environment and virtualize it, it's easy to set up the same environment across multiple instances

There are vital factors that need to be considered when using Cloud Run in Google Cloud Platform that is: the computing cost of the services, the service Resource (CPU, Ram) requirement, how frequently it will be accessed, and how it handle shared data.

REFERENCES

[1] WHO, Basic documents, 43rd Edition. Geneva: World Health Organization, 2014.

[2] Indonesia National Adolescent Mental Health Survey (I-NAMHS), 2023, [Online]. Available: https://qcmhr.org/~teoqngfi/outputs/reports/12-i-namhs-report-bahasa-indonesia/file.

[3] C. Boettiger, “An introduction to Docker for reproducible research,” in *Operating Systems Review (ACM)*, 2015. doi: 10.1145/2723872.2723882.

[4] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: State-of-the-art and research challenges,” *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, May 2010, doi: 10.1007/s13174-010-0007-6

[5] S. Singh and N. Singh, “Containers & Docker: Emerging roles & future of Cloud technology,” in *Proceedings of the 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology, iCATccT 2016*, 2017. doi: 10.1109/ICATCCT.2016.7912109.

[6] M. Fadlulloh and R. Bik, “IMPLEMENTASI DOCKER UNTUK PENGELOLAAN BANYAK APLIKASI WEB (Studi Kasus : Jurusan Teknik Informatika UNESA),” 2017.

[7] K. Relan, *Building REST APIs with Flask*. 2019. doi: 10.1007/978-1-4842-5022-8.

[8] B. B. Rad, H. J. Bhatti, and M. Ahmadi, “An Introduction to Docker and Analysis of its Performance,” *IJCSNS International Journal of Computer Science and Network Security*, vol. 17, no. 3, 2017, [Online]. Available: https://www.researchgate.net/publication/318816158

[9] J. Shah and D. Dubaria, “Building modern clouds: Using docker, kubernetes google cloud platform,” in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference, CCWC 2019*, 2019. doi: 10.1109/CCWC.2019.8666479.

[10] Google, “Google Cloud Products.” https://cloud.google.com/products (accessed Oct. 21, 2022).

[11] Google, “Overview of Cloud Build.” https://cloud.google.com/build/docs/overview (accessed Oct. 21, 2022).

[12] I. Barokah and A. Asriyanik, “Analisis Perbandingan Serverless Computing Pada Google Cloud Platform,” *Jurnal Teknologi Informatika dan Komputer*, vol. 7, no. 2, 2021, doi: 10.37012/jtik.v7i2.662.

[13] C. Khawas and P. Shah, “Application of Firebase in Android App Development-A Study,” *Int J Comput Appl*, vol. 179, no. 46, pp. 49–53, Jun. 2018, doi: 10.5120/ijca2018917200.

[14] Google, “Cloud Firestore Documentation.” https://firebase.google.com/docs/firestores (accessed Oct. 22, 2022).