Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 6, Number 3, July 2022
DOI : https://doi.org/10.33395/sinkron.v7i3.11522
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Api Service Infrastructure using Kubernetes and Terraform Based nn Microservices Ngoorder.id

Azwar Riza Habibi[1], Galih Laksono[2],
[1)2)] Institut Teknologi dan Bisnis Asia Malang, Indonesia
[1)]riza.bj@gmail.com, [2)]galihlaksono98@gmail.com

Submitted : 5 July 2022 | **Accepted** : July 21, 2022 | **Published** : July 30, 2022

Abstract: The growing number of ngorder.id service users causes traffic to Api Ngorder to be higher, so a new infrastructure is needed in order to maintain Api Ngorder uptime during high traffic and also maintain service stability. In the implementation process, Api scripts that are currently running on a monolith cluster will be divided into several categories and will be split into several kubernetes clusters. To support autoscale, a Horizontal Pod Autoscaler was added, and to route traffic it would use the Api Gateway from Amazon Web Service. In this infrastructure test, it is done by testing the logic script function using Katalon Studio and testing at the infrastructure level by doing a crash test in the form of failing to deploy and terminating the pod, as well as performing a stress test to test autoscaling in the cluster, the entire test can be run by performing a stress test on the php service pods. by setting the autoscaler parameter Memory Utilization Percentage 125%, 150% and 250%, proving that the HorizontalPodAutoscaler (HPA) as an autoscaler handler can function according to the targets and parameters that have been determined.

Keywords: Infrastructure, Api Service, API Gateway, Microservice, Ngorder

INTRODUCTION

Ngorder.id is a technology company engaged in providing online store platforms and store management that can connect many marketplaces in one dashboard, app.ngorder.id, besides that, app.ngorder.id also has versions of applications based on Android and IOS. With the many services and platforms it has, Ngorder.id also cooperates with various parties to support the functionality and diversity of services in its products, in communication between Ngorder.id services and also third parties who collaborate with Ngorder using the API Service (de O. Junior, et al., 2022).

One example of the application of the API Service on ngorder.id is as a communication interface between services provided by ngorder and the ngorder mobile application for the needs of users' online stores, the Ngorder API itself is published to its users so that developers from users can also integrate existing applications with the service. order.id.

The addition of kuberneter to the ngorde.id system to manage application workloads as well as configuration and automation (Hu & Wang, 2021). It is used to optimize operating systems that are capable of running applications across various clusters and infrastructure in cloud services and private data center environments. This will certainly have an impact on the quality of ngorder.id services (Liu, Haihong, & Song, 2020).

However, along with the growing number of ngorder.id users and also third parties integrating into ngorder.id, the number of loads served by the Ngorder API server is also increasing, so a reliable, secure and effective infrastructure is needed to withstand incoming loads without any loss of quality (Wan, Wu, & Zhang, 2020). service. development of the Ngorder API infrastructure, which currently uses the load balancing system in AWS' elastic beanstalk service (Amazon Web Services) to a more reliable system based on API Gateway and microservices (Dragoni, et al., 2017). However, the implementation using the ngorder.id load balancing system still finds excessive load buildup, so that the services provided by ordering are decreasing in the load time provided by the system (Daya, et al., 2015).

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 6, Number 3, July 2022
DOI : https://doi.org/10.33395/sinkron.v7i3.11522
e-ISSN : 2541-2019
p-ISSN : 2541-044X

LITERATURE REVIEW

Monolithic architecture

Monolithic architecture is an architecture that describes an application that runs all logic on a single application server (Al-Debagy & Martinek, 2020). Applications that are built with a monolithic architecture are only run using one application server, so they only require the application maintenance process on one application server (Mufrizal & Indarti, 2019). In the research, it is explained that monolithic architecture is a software application whose modules cannot be run independently. By using a monolithic architecture, each application is run concurrently because all application modules are contained in a very large application (Filippone, Autili, Rossi, & Tivoli, 2021). There are several disadvantages of monolithic architecture, namely:

1. Large in size so it is very difficult to perform maintenance and application development.
2. There can be library dependency conflicts during the process of adding new modules.
3. Each addition of a new module requires a restart of the application.
4. The emergence of application code conflicts when the application is deployed simultaneously.
5. Monolithic architecture has limited scalability.
6. Using only one technology to create applications using a monolithic architecture.

Microservice architecture

Microservice architecture is a software architecture style that requires solving business applications, while the research explains that microservice architecture is a new architecture where application development is carried out in the form of small web services that communicate with each other (Bushong, Das, Al Maruf, & Cerny, 2021). There are several advantages of microservice architecture (Schmidt & Hristovski, 2016), namely:

1. Microservice architecture makes application code less independent and therefore can be tested independently of the application.
2. Easy software maintenance.
3. Can perform the software distribution process independently.
4. Easy to do scalability.
5. Developers can freely develop applications with various programming languages and frameworks.

Terraform

Terraform is a platform for creating, modifying and combining infrastructure safely and efficiently using code, this concept is known as Infrastructure as a Code or IaaC (Dinh-Tuan, Katsarou, & Herbke, 2021). With Terraform, it is possible to automate infrastructure management as well as do configuration documentation. According to the explanation in the documentation, Terraform itself already supports many platforms such as Amazon Web Services, Google Cloud Platform, Azure, Kubernetes and many more, so DevOps Engineers can perform multiplatform infrastructure management more easily (Liu K. , 2021).

Kubernetes

Kubernetes is an open-source platform that functions for the management of containerized application workloads, also provides a set of declarative configurations and optimizations (Ding, Wang, & Jiang, 2022). In its own implementation, Kubernetes requires several main components to run, which are categorized into Master Nodes and Worker Nodes (Dewi, Noertjahyana, Palit, & Yedutun, 2019).

System ngorder.id

In order to support large workloads, the Ngorder API system infrastructure already uses load balancing and auto healing technology provided in Amazon Web Services' Elastic Beanstalk service as the cloud provider that is currently being used, here is the infrastructure scheme currently used:

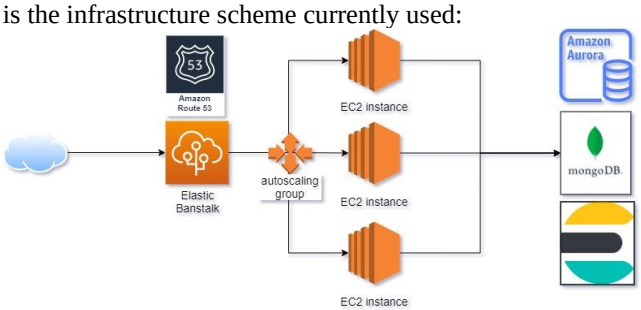

Fig. 1 Current infrastructure

The current infrastructure uses:

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 6, Number 3, July 2022
DOI : https://doi.org/10.33395/sinkron.v7i3.11522
e-ISSN : 2541-2019
p-ISSN : 2541-044X

1. Route 53 as a DNS management platform.
2. Elastic Beanstalk as a clustering platform that already has load balancing, auto healing and provisioning features if a developer deploys the latest API version. Elastic Beanstalk also has a function as a traffic divider just like a load balancer function.
3. Auto scaling group functions as a configuration site and EC2 server template that will be triggered by elastic beanstalk to scale out or scale in.
4. EC2 Instance as a deployment server that contains scripts and also a webserver for API purposes.

Even though auto scaling and load balancers have been implemented, in practice the current infrastructure is not yet capable of serving the needs of the ngorder.id service as a whole, to find out the relationship between services, a connection diagram between services is made as follows:

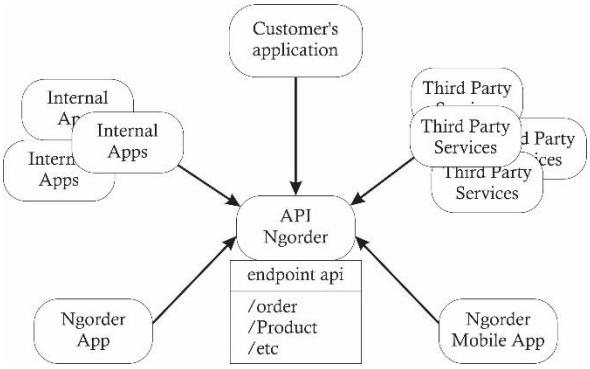

Fig. 2 Relationships between services

From fig 1 and fig. 2 we can conclude that currently all ordering services are still using a centralized API, so that the load on each server remains high even though it is already using load balancing. From the analysis of the current infrastructure, it can be concluded that currently it still has the following weaknesses:

1. Have not implemented a separate system between API categories.
2. Scaling is still done on the server side (all endpoints), so the scale out process takes longer.
3. The cluster system is still tied to each other for the distribution of traffic.

Ngorder.id API service already uses a cluster system using elastic beanstalk, but because it is still a cluster server, the server scale up process still takes a long time.

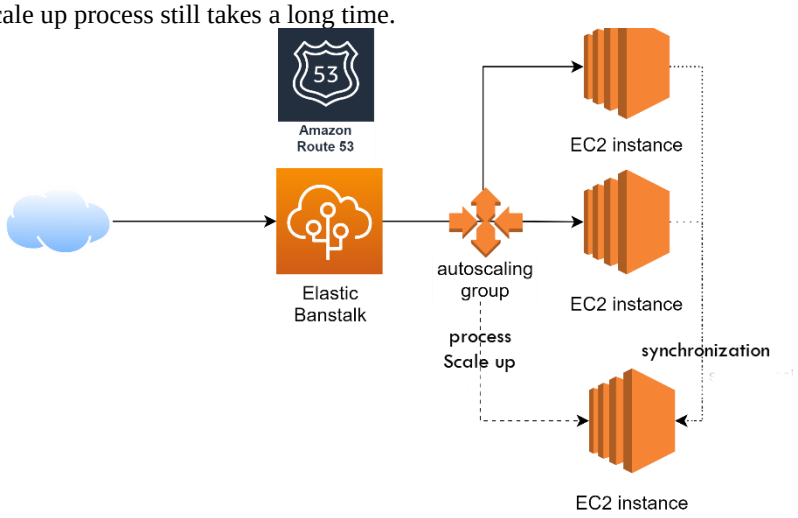

fig. 3 *flow scale up*

on fig. 3 explains how the scaling process is carried out, after receiving a scale up request from Elastic Beanstalk, Autoscaler will clone EC2 from the template that has been set in the autoscaling group, then the next service, library and other needs that have been defined will be prepared. After all processes are complete, then

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 6, Number 3, July 2022
DOI : https://doi.org/10.33395/sinkron.v7i3.11522
e-ISSN : 2541-2019
p-ISSN : 2541-044X

EC2 is ready to receive traffic, because of the length of this process, the scale up process can take up to more than 5 minutes.

METHOD

At this stage, the system design and implementation planning will be carried out to resolve the weaknesses of the current infrastructure. From the system side and application flow, the modeling can be seen as below:

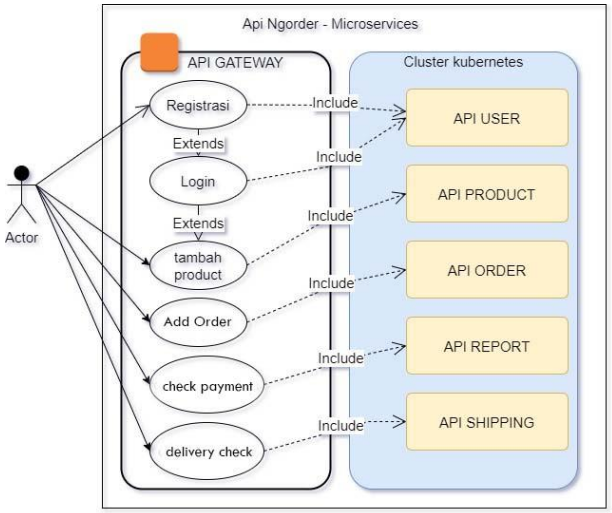

Fig. 4 Use Case Diagrams

As explained by the use case diagram that each process has a separate category and API service from each other, and the Api gateway task will route each API service endpoint according to the path parameters specified in each API process.

At this stage, a system design or mapping will be carried out from the results of the previous categorization. The following is an overview of the infrastructure to be built:

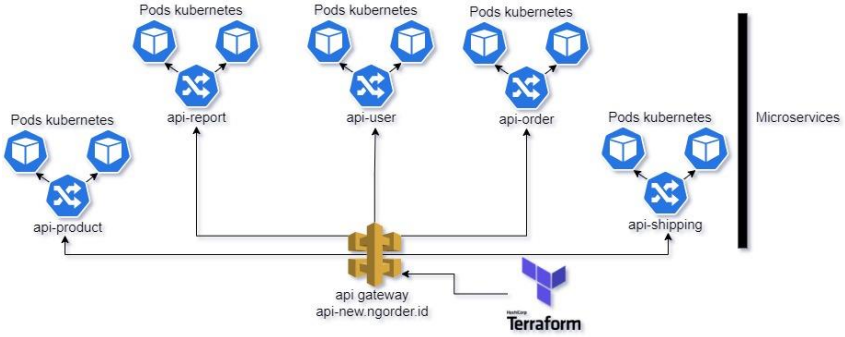

Fig. 5 new infrastructure Ngorder

As fig explains. 5 every incoming request will be directed first at the api gateway, then it will be routed based on the access path, for example if you want to hit/access https://api-new.ngorder.id/api/open/product then the api gateway will be directed to ingress in the api-product namespace to process incoming requests. In addition, with product APIs that are isolated from other APIs, it will provide benefits for developers if they decide to add additional services but only run on product APIs, such as adding worker supervisors, so workers will run on product API services only.

With a separate and specific system like this, allowing the operational team to implement autohealing and isolation if there is a disruption in one API service without affecting other services, autohealing in Kubernetes itself is available using the livenesprobe or readinesprobe parameters which will terminate the problematic service and reconnect. create the problematic pod quickly.

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 6, Number 3, July 2022
DOI : https://doi.org/10.33395/sinkron.v7i3.11522
e-ISSN : 2541-2019
p-ISSN : 2541-044X

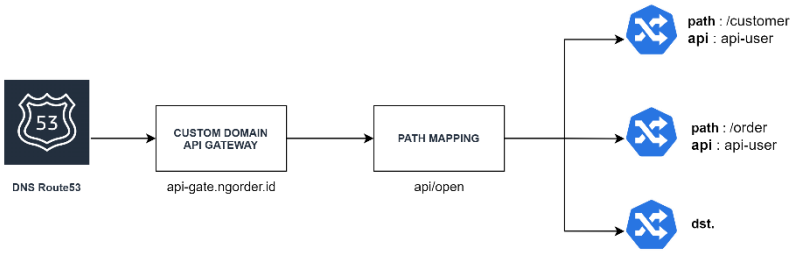

Fig. 6 Api Gateway

As fig explains. 6, the configuration on the Api Gateway side consists of 3 parts:

1. Route53, is used as a DNS server so that the API can be accessed with the domain belonging to ngorder.id.
2. Custom Domain, functions as a substitute for the domain that is automatically generated from the Api Gateway to the api-gate.ngorder.id domain.
3. Path Mapping, used to map from custom domain to url from target.

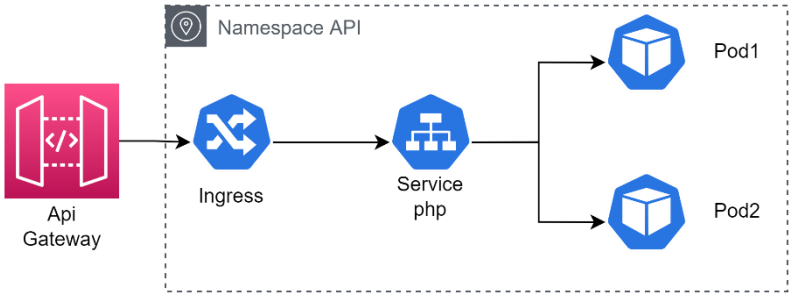

Fig. 7 Kubernetes Architecture

As shown in Fig. 7, each Namespace will consist of several parts, namely:

1. Ingress, functions as a regulator of external access to the Service and the location of the host definition in the namespace.
2. Service, serves to direct traffic into each Pods that are within the Service, in this architecture the Service is used to manage and share traffic that goes to the PHP Pod, where the Api Ngorder script is located.
3. Pod, as the container for the Image Docker location of the Ngorder Api to run.
4. Namespace, serves as a cluster separator between API categories

This division allows each API to have the freedom to add new services according to the needs of each API without having to change the infrastructure of other APIs.

After the deployment is complete, Kubernetes components will be formed to run the API services that have been created, such as ConfigMap, Secret, Ingress, Pods PHP and also Pods Supervisor to run workers. Fig. 8 is an example of a component that is deployed for Api-order and the correlation between components is displayed in realtime by the Kubernetes dashboard.

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 6, Number 3, July 2022
DOI : https://doi.org/10.33395/sinkron.v7i3.11522
e-ISSN : 2541-2019
p-ISSN : 2541-044X

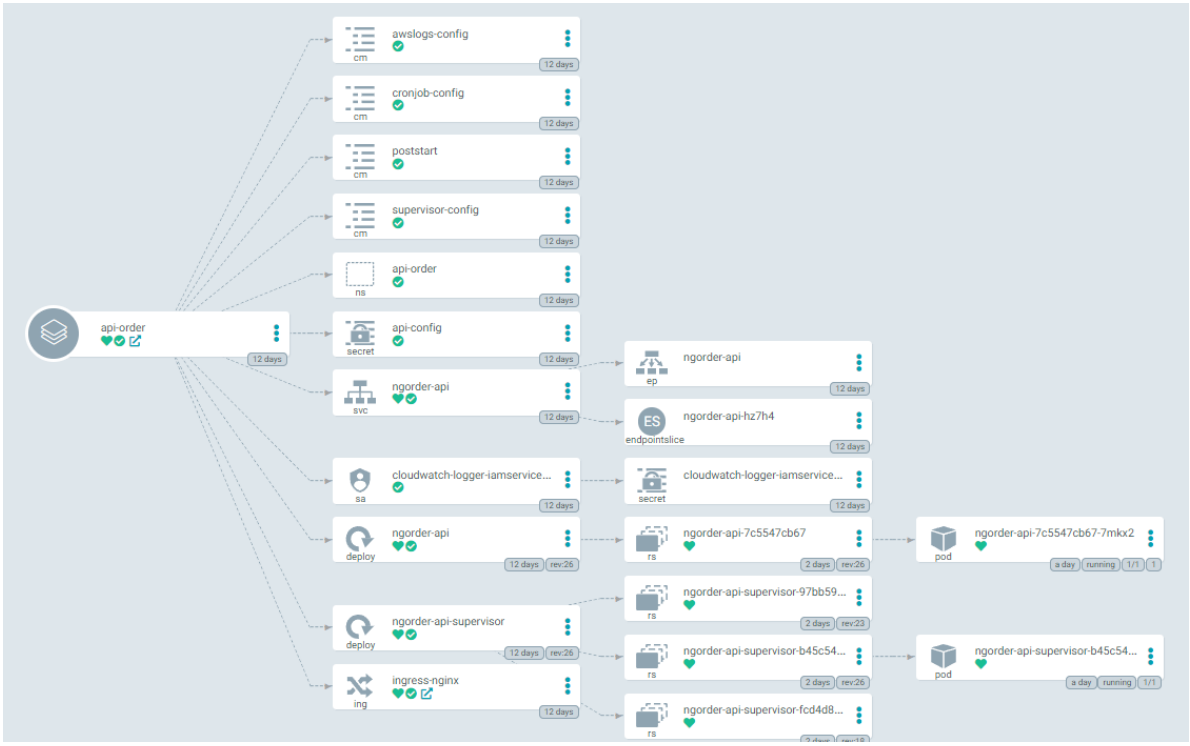

Fig. 8 *Mapping Deployment Api-order*

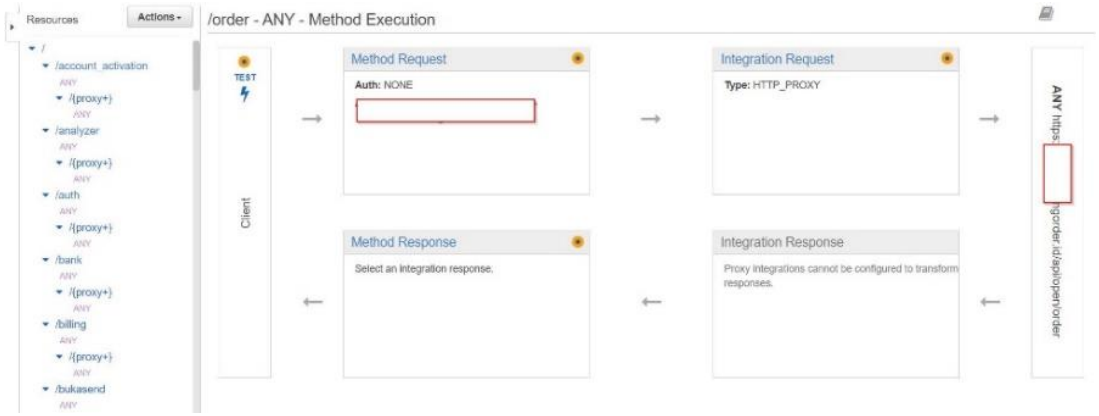

Fig. 9 Dashboard Api Gateway

In this test, we will use an automated testing script using Katalon Studio which has been made by the Quality Assurance Engineer team, this test serves to prove the new infrastructure and the api script can run normally for system needs, in this test we will use an api-order including the create process. order, selection of delivery services and customer addresses, delivery status updates, payment status updates and payment confirmations.

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 6, Number 3, July 2022
DOI : https://doi.org/10.33395/sinkron.v7i3.11522
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Fig. 10 Katalon Studio Test Dashboard

From the test results fig. 10 above shows that the functions in the create order process are appropriate and functioning properly without any errors that appear.

RESULT

The test design uses a managed service Kubernetes from Amazon Web Service (AWS) called Elastic Kubernetes Service (EKS) with the following specifications.

Table 2 Kubernetes software testing requirements

No	Specification	Description
1	Versi Kubernetes	1.19
2	Cluster Instance	R5.Large
3	vCPU	2 vCPU @3.1 GHz
4	Memory	16 GiB
5	Bandwith	Up to 10Gbps

The following stress test pods will send wget requests continuously to http://ngorder-api (the service address for the internal namespace), while the autoscaler itself will try to automatically adjust the average memory usage in the php service pod below the target by scaling out if the average memory usage has exceeded the parameters specified in the HorizontalPodAutoscaler (HPA), here are the test results with different HPA parameters:

Table 2 Autoscaling Test Results

No	Aktifitas	Target	Hasil	Keterangan
1	Change the HPA memory utilization target to 125% and start 5 pod stress tests.	Pod Api-order *scale out*	Succeed	When memory usage exceeds 125%, HPA adds 1 additional php pod.
2	Change the HPA memory utilization target to 150% and start 10 pod stress tests.	Pod Api-order *scale out*	Succeed	When memory usage exceeds 150%, HPA adds 1 additional php pod.
3	Change the HPA memory utilization target to 250% and start 20 pod stress tests.	Pod Api-order *scale out*	Succeed	When memory usage exceeds 250%, HPA adds 1 to the extra php pod.

Based on the results of the autoscaling test in Table 1, it proves that the Horizontal Pod Autoscaler (HPA) as an autoscaler handler can function according to the targets and parameters that have been determined. The autoscaler system can scale up and provide a response according to the test plan and design that has been prepared.

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 6, Number 3, July 2022
DOI : https://doi.org/10.33395/sinkron.v7i3.11522
e-ISSN : 2541-2019
p-ISSN : 2541-044X

In this benchmarking process, wrk will be run which functions to benchmark by sending HTTP requests based on predetermined parameters, namely 12 threads, 30 seconds and 400 connections, here are the results:

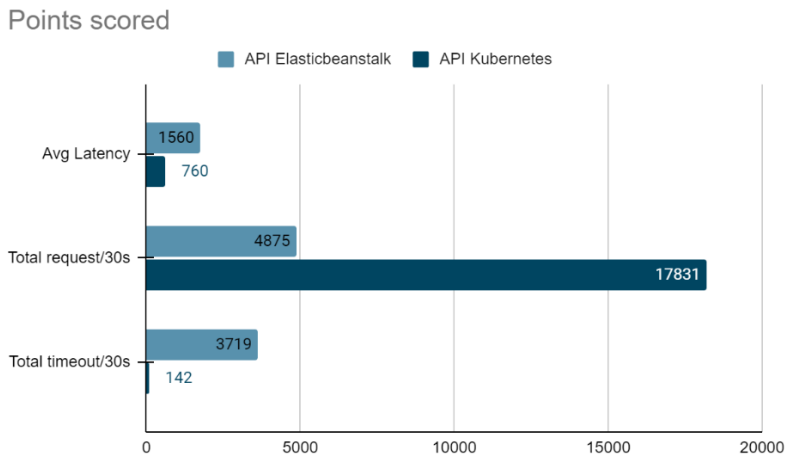

Fig. 11 Visualization of benchmark results

In Fig. 11 shows that with the new infrastructure, namely the Kubernetes API label, it excels in a low average latency of 760ms, besides the total requests that can be received in 30 seconds are higher at 1783 requests/30 seconds with a low total timeout, namely 142 request timeouts.

DISCUSSIONS

Given the limited tools and other operational constraints in this study, the GitOps principle was not applied, so researchers could improve it by applying the GitOps principle at the time of deployment. Performance of the system This infrastructure only uses Amazon Web Service, but to improve the quality of service to use multi cloud providers in Kubernetes clusters and add a service mesh architecture.

CONCLUSION

The results of implementing the Ngorder API to the microservice system can be said that by using Katalon Studio, it can be said that Amazon Web Service's API Gateway service can properly distribute traffic into Kubernetes clusters according to the specified path categories. Based on the tests carried out, it can also be said that the logic functions in the Ngorder API script do not experience problems when transferred to infrastructure with a microservice system. The results of the autohealing and autoscaling test Kubernetes Clusters are designed to autoscaling in less than 10 seconds when they reach the limit point of their resource usage. This shows that the Kubernetes API excels in the average latency of 760ms, besides the total requests that can be received for 30 seconds are higher at 1783 requests/30 seconds with a total timeout of 142 request timeouts.

REFERENCES

Al-Debagy, O., & Martinek, P. (2020). Extracting Microservices' Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach. *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)*, 289-294.

Bushong, V., Das, D., Al Maruf, A., & Cerny, T. (2021). Using Static Analysis to Address Microservice Architecture Reconstruction. *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 1199-1201.

Daya, S., Van, N., Kameswara, D., Carlos, E., Ferreira, M., Glozic, D., . . . Vennam, R. (2015). *Redbooks Microservices from Theory to Practice Creating Applications in IBM Bluemix Using the Microservices Approach.* U.S.

de O. Junior, R., da Silva, R., Santos, M., Albuquerque, D., Almeida, H., & Santos, D. (2022). An Extensible and Secure Architecture based on Microservices. *2022 IEEE International Conference on Consumer Electronics (ICCE)*, 1-2.

Dewi, L., Noertjahyana, A., Palit, H., & Yedutun, K. (2019). Server Scalability Using Kubernetes. *2019 4th Technology Innovation Management and Engineering Science International Conference (TIMES-iCON)*, 1-4.

Ding, Z., Wang, S., & Jiang, C. (2022). Kubernetes-Oriented Microservice Placement with Dynamic Resource Allocation. *IEEE Transactions on Cloud Computing*, 1.

*name of corresponding author

e-ISSN : 2541-2019
p-ISSN : 2541-044X

Dinh-Tuan, H., Katsarou, K., & Herbke, P. (2021). Optimizing microservices with hyperparameter optimization. *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*, 685-686.

Dragoni, N., Giallorenzo, S., Lafuente, A., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017, 11). Microservices: Yesterday, today, and tomorrow. *Present and Ulterior Software Engineering*, 195-216.

Filippone, G., Autili, M., Rossi, F., & Tivoli, M. (2021). Migration of Monoliths through the Synthesis of Microservices using Combinatorial Optimization. *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 144-147.

Hu, T., & Wang, Y. (2021). A Kubernetes Autoscaler Based on Pod Replicas Prediction. *2021 Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS)*, 238-241.

Liu, K. (2021). *Microservice Migration Patterns and how Continuous Integration and Continuous Delivery are affected A Case study of Indicio's journey towards microservice.* SWEDEN.

Liu, Q., Haihong, E., & Song, M. (2020). The Design of Multi-Metric Load Balancer for Kubernetes. *2020 International Conference on Inventive Computation Technologies (ICICT)*, 1114-1117.

Mufrizal, R., & Indarti, D. (2019, 4). Refactoring Arsitektur Microservice Pada Aplikasi Absensi PT. Graha Usaha Teknik. *Jurnal Nasional Teknologi dan Sistem Informasi, 5*(1), 57-68.

Schmidt, A., & Hristovski, D. (2016). *A Simplified Database Pattern for the Microservice Architecture.* IARIA.

Ueda, T., Nakaike, T., & Ohara, M. (2016). Workload characterization for microservices. *2016 IEEE International Symposium on Workload Characterization (IISWC)*, 1-10.

Wan, F., Wu, X., & Zhang, Q. (2020). Chain-Oriented Load Balancing in Microservice System. *2020 World Conference on Computing and Communication Technologies (WCCCT)*, 10-14.

Zhu, M., Kang, R., He, F., & Oki, E. (2021). Implementation of Backup Resource Management Controller for Reliable Function Allocation in Kubernetes. *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, 360-362.

*name of corresponding author