OIC-CERT Journal of Cyber Security
Volume 4, Issue 1 (April 2022)
97 - 110

Building A Dynamic Cloud-Based Snort NIDS: A Journey of "Mata Garuda" Network Intrusion Monitoring Development in Indonesia

Ferry Astika S.[1], Fadhil Yori[2], Ikbar Maulana[3], Dimas R[4], Ahmada Y[5], M. Alfiyan[6], Andri S[7], Novi Turniawati[8], Dani Ramdani[9], Taufik Y[10], Muhammad Salman[11], Kalamullah Ramli[12], Jauhari[13],Isbat[14], Iwan Syarif[15]

[1,11,12]Electrical Engineering, Universitas Indonesia, Depok, Indonesia
[2,3,4,5,6,13,14,15] Politeknik Elektronika Negeri Surabaya, Indonesia
[7,8,9,10] Balai Jaringan Informasi Informasi dan Komunikasi, Badan Pengkajian dan Penerapan Teknologi, Jakarta, Indonesia

{[1]ferry.astika, [10]muhammad.salman, [11]kalamullah.ramli}@ui.ac.id,
{[2]fadhilyori,[3]ikbarmaulana77,[4]dimasrizkyhp,[5]ahmadayusril,[6]alfiyan}@it.student.pens.ac.id
{[7]andri.saputra,[8]novi.turniawati,[9]dani.ramdani,
[10]taufik.yuniantoro}@bppt.go.id

ARTICLE INFO

Article History
Received 8 Feb 2022
Received in revised form 12 Apr 2022
Accepted 15 Apr 2022

Keywords:
Keywords-component; snort; big data; cloud-based IDS; lambda architecture

ABSTRACT

Abstract—Snort is one of the well-known signature-based network intrusion detection system (NIDS). In the typical NIDS architecture, the sensor placement must be in the same physical network and the defence centre that makes the deployment cost steep. The increasing number of sensor instances, followed by a rapid increase in log data volume, caused the existing system to face big data challenges. Snort must have an efficient mechanism to collect, store, and aggregate data to address this problem. In this research, we want to fulfil the demands faced by Snort. We propose a new analysis framework for Snort NIDS on cloud and big data technology. Using our proposed framework, we can reduce deployment costs of NIDS, which run on big data environments. It contains Docker as the sensor's platform, Apache Kafka as the distributed messaging system, Apache Spark as the distributed processing engine, and

ISSN 2636-9680
eISSN 2682-9266

Apache Cassandra as the core databases. Experiments are conducted to measure sensor deployment and aggregation speed and efficiency and data processing performance efficiency. As a result, our proposed framework requires a shorter deployment time of the Snort sensor and a lower system deployment cost. The data storing and aggregation are faster and more efficient than the typical architecture of Snort NIDS.

I. INTRODUCTION

Snort is one of the commonly used signature-based NIDS[1]. We can find various implementations of Snort in many network security systems. Typically, users install the Snort sensors to detect intrusion in their networks. Then the set of sensors send the log data to a dedicated defence centre (DC). DC is responsible for processing and aggregating the data. The physical placement of sensors and defence centre must be on the same local network in the typical architecture. With the typical architecture, installing sensors at a distant DC location without re-deploying another defence centre is impossible. The placement of sensors and a defence centre must be in the same local network.

One of Snort NIDS's best practices was on Mata Garuda Project under the Indonesia Security Incident Response Team on Internet Infrastructure/Coordination Center (IDSIRTII/CC)[2]. In 2014, IDSIRTII/CC installed Snort-based NIDS sensors on twelve ISP routers that handle most Indonesia Internet traffics. Intrusion detection and data aggregation Sensors and the DC are located in the same networks. The sensor has two interfaces, one for sniffing the packet, and the other interface sends the intrusion logs to DC through a secure file transfer protocol every minute. Then in DC, we aggregate the data in various time units, enriching the data with IP geolocation for building attack maps and other security-related analyses.

On the other side, emerging cloud technology has become more prevalent in society. Monitored servers and networks by an administrator can be anywhere. In this situation, we cannot use the typical architecture of NIDS as figured by Mata Garuda. The dynamic change in cloud-based architecture requires ubiquitous sensor placement and lightweight sensor deployment and provides the best data transport from DC agents as reliable. Moreover, cloud technology usage causes a significant increase in data volume on the defence centre side. With the

rise in the volume, velocity, and variety of data that must be processed, the defence centre certainly needs a big data platform to overcome it [3], [4],[5].
Our method aims twofold in this paper:

- We propose a novel scheme to implement Snort-based NIDS for the cloud against intrusion by using docker container technologies and IoT architecture.
- We describe the components, architecture, and components of the proposed design in detail.
- We propose a new design for the defence centre to handle massive data from a set of sensors over the clouds by using the advantages of big data technologies.

This paper has been structured as follows: Section II reviews related works in NIDS technology in a real network and implements big data technology for NIDS. We describe our proposed design in Section III to build dynamic and scalable parallel Cloud-Based Snort NIDS using containers and big data.

II. Related Works

Back to 2014 when the IDSIRTII/CC developed a Snort based NIDS to monitor network intrusion over leading ISPs in Indonesia. The system consists of twelve Snort NIDS. Each Snort NIDS, namely a sensor. It sniffs the traffic from ISP's router. The sensor detects and logs the intrusion based on its previously configured rules. Then the sensors send their logs to DC through a secure file transfer protocol every minute. The DC aggregates the logs according to the time unit from minute to year. It maps each record's IP source and destination to their location through IP2Maps DB. Figure 1 shows the topology of Mata Garuda implemented at IDSIRTII/CC.

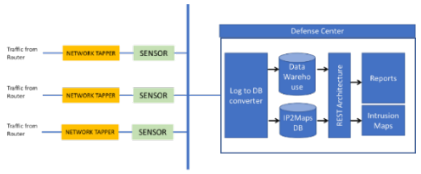

Fig. 1. The topology of Mata Garuda, Indonesia intrusion detection system based on Snort NIDS at IDSIRTII/CC.

In early 2015, When implementing Mata Garuda in a real gigabit network, we found that our system can only efficiently handle 4-5 million data per query. As the data grow exponentially, the data table also expands in its size. Many join queries over several large tables cause Mata Garuda to run slowly. Another problem is that the topology used still uses only one OLTP database server, so the server load is high. We propose a solution using a partition table design as a database design for Mata Garuda. The result is that the computation time is much faster and improves Mata Garuda's performance in handling extensive data[6].

In early 2020, the rapid change of Internet technology with cloud

ISSN 2636-9680
eISSN 2682-9266

computing and big data technology in Indonesia challenged us to develop a new version of Mata Garuda. The improved version of Mata Garuda must be compatible with cloud technology. Besides that challenge, along with the increasing number of sensor instances followed by a rapid increase in log data volume, caused the existing system to face big data challenges. Based on our study in [7] and [8], we successfully process (ETL and data enrichment) snort log files using the Big Data principle and data mining in the Mata Garuda application. The data mining method is carried out on geolocation data to get the attack's location with our proposed distributed system. We used the SQL-UDTF (User Defined Table Generating Function) feature in Hadoop to perform and compare them with join queries. The algorithm applied in the mining process is *k*-means clustering to obtain clusters from GeoIP attacks. As a result, UDTF can reduce the computation time to 0.08 seconds, which initially took 3561 seconds using join queries. Figure 2 shows the architecture of an improved version of Mata Garuda implementing big data technology.

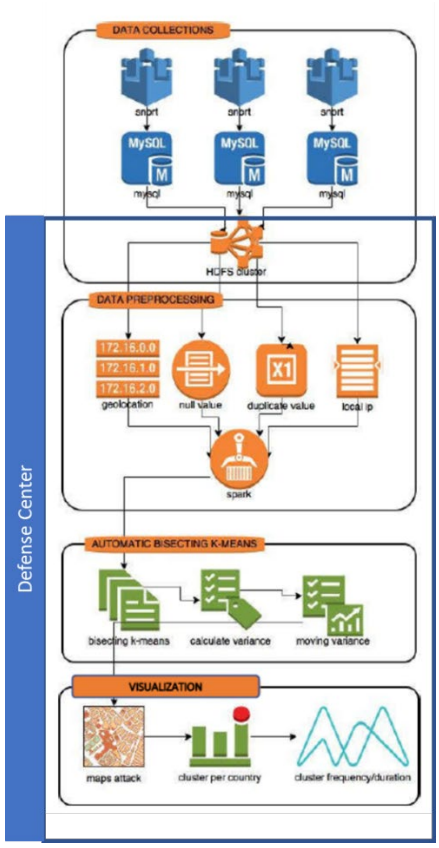

Fig. 2. The improved version of Mata Garuda implementing big data technology.

Conceptually similar work, utilizing a similar method, was proposed by [9], [10], and [11]. In this literature, the researchers have extensively investigated big data technology implementation to improve NIDS performance. They used offline well-known IDS datasets to conducts the experiments. The data set is divided into two parts: the training data set and the testing dataset. They use the training dataset processed by various machine learning algorithms implemented in the big data environment to create a model. Then the model is used as the rule of

the detection engine to detect intrusions. Then they stream the testing data to the detection engine and measure the performance by using the accuracy, precision, and various machine learning evaluation metrics.

More realistic implementation of big data in the NIDS system is introduced by BigFlow [12]. The BigFlow system consists of five main parts: monitored agents, message middleware, stream processing parts, stream learning parts, and analytics. The agent sends the network event to the messaging middleware, which acts as a broker of events. The event then streamed to the stream processing line to extract 158 features from its bidirectional network flow. The stream learning module processes the features from the captured flow in 15 seconds time window to create an initial classification model. Moreover, the stream learning module provides reliable classifications, employing a Verification module; at the same time, it provides updated ML models. Regarding the authors, BigFlow can maintain high accuracy over a network traffic dataset spanning a full year.

Most of the current evidence supports multi-agents, messaging middleware, big data, and machine learning methods in NIDS. It means the complexity of the entire NIDS increase. The complexity means difficulties in arranging different parts connected in a complicated way while keeping the system configuration and maintenance as simple as possible.

Unfortunately, there may not be sufficient relevant literature for using Snort as the core of the distributed agents in the cloud and big data environment. This section briefly discusses the containerization technology in IDS for deploying Snort NIDS sensors, selecting the suitable message middleware system to provide low latency in data transmission, the Hadoop platform, and the lambda architecture design building real-time processing.

A. The docker containerization technology

Docker is an open-source platform used by development teams to effectively build, run, and distribute applications built together [13]. Docker technology consists of two elements, namely the Docker Engine and the Docker hub. The Docker engine is a portable software packaging tool with a lightweight system through a particular library, namely *libcontainer* (see figure 3). Using this library, Docker can manipulate namespaces, control groups, SELinux policies, network interfaces, and firewall rules. This feature allows independent containers to run within a single instance, so the overhead of starting virtual machines can be avoided. The second part is the Docker Hub. It is a Docker application sharing on a cloud service, including its

ISSN 2636-9680
eISSN 2682-9266

workflow automation. Creating new containers is easy. So, it enables us to serve the rapid iteration of applications and provide transparency for application updates.

Moreover, using Docker, we can reduce the development phase, testing, and production time. These advantages lead us to build our Snort-based IDS sensor based on Docker technology[14]. We can build the Snort-based NIDS and its dependencies into one docker image. The users can download this pre-configured sensor application without any further configuration, as simple as issuing the Command-Line Interface (CLI) "docker run." (see figure 4)

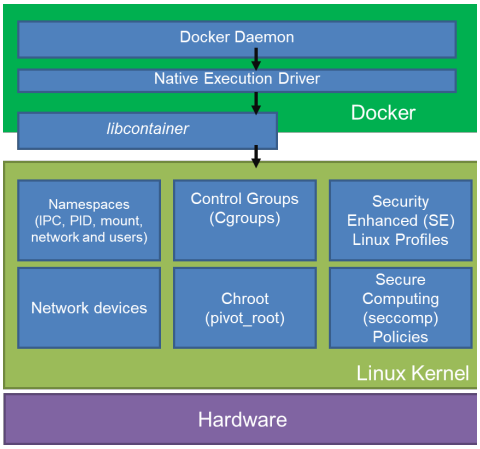

Fig. 3. Docker uses *libcontainer* to access the Linux Kernel.

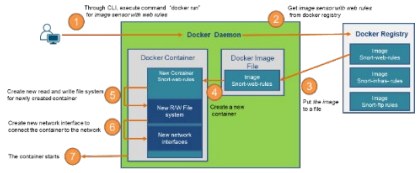

Fig. 4. A process for creating a new docker container of the sensor (Snort based NIDS with pre-configured web rules)

B. The messaging middleware system

Messaging middleware offers a hub and spoke architecture that serves as a central point of communication between all applications. It controls the transport method, the rule, and the data reformatting for ensuring the data arrives at the receiving application precisely. For example, when data are sent by one application (publisher), they can be stored in a queue and then forwarded to the receiving application (subscriber) whenever it becomes available to process. Commonly, the messaging middleware system is implemented in IoT architecture. Most IoT projects combine MQTT and Kafka for performance and scalability reasons. The high-level architecture is shown in figure 5.

MQTT Broker receives data from sensors (as publisher) by using MQTT protocols. It sends to Apache Kafka Broker through the MQTT connector. The Kafka Producer is an application that publishes data to a Kafka cluster made up of Kafka brokers. The Kafka Broker will be responsible for receiving and storing the data

when a producer publishes, namely the recent data from MQTT Broker. Then, the Kafka Consumer consumes data from Kafka broker at a specified offset, i.e., timestamps, position. Each consumer can do a specific task, i.e., write the messages to a Hadoop or stream the Spark engine's data. A basic unit of data in Kafka is called a message. It contains the data and its metadata, i.e., timestamps, type of data compression. These messages are organized into logical groupings called topics, on which producers publish data. Typically, messages in a topic are distributed across different partitions in different Kafka brokers. A Kafka broker can manage many partitions.

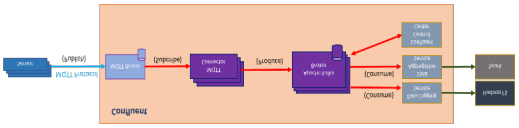

Fig. 5. The high-level architecture of a typical IoT system by combining MQTT and Apache Kafka. Confluent manages all elements in MQTT and Apache Kafka.

C. The Hadoop platform and the lambda architecture

Hadoop is an open-source platform designed for storing and processing an extensive amount of data. It relies on distributed hardware to store and process data, enabling large processing amounts of data on distributed clusters of servers. The Hadoop consists of a storage component, namely Hadoop's Distributed File System (HDFS), and a processing component called MapReduce[15]. HDFS is a distributed storage file system that is designed to run on commodity hardware. An HDFS cluster consists of NameNode(s) that manage the file system metadata and many DataNodes that store data. A file is split into blocks, and these blocks are stored in a set of DataNodes. Each block has several replications distributed in different DataNodes. MapReduce is the processing component in the Hadoop platform. It mainly consists of JobTracker as master nodes and as slave nodes per cluster. The JobTracker is responsible for scheduling jobs for TaskTrackers, monitoring them, and re-executing the failed tasks. The MapReduce and HDFS run on the same set of nodes.

The Lambda architecture's basis is to compute arbitrary functions on distributed datasets in real-time and combining batch and real-time processing capabilities to balance data latency throughput and fault tolerance. However, there is no optimal design that can accomplish this task. Instead, several tools and designs are used to build a complete big data system. The Lambda architecture addresses the problem of arbitrary computing functions parallel to distributed data in real-time. It consists of a three-layered architecture: batch, speed, and serving layers (see figure 6).

1) The Batch Layer

   It has two tasks. The first is to store the continually growing and immutable master data in a distributed file system, a Hadoop distributed file system

ISSN 2636-9680
eISSN 2682-9266

(HDFS). The second task is to precompute batch views for this distributed data by using MapReduce. Those batch views can be used to reply to incoming queries with low read latency.

2) The Speed Layer
Comparing to the batch layer, it does not precompute the views for the entire data. It also has two tasks. The first is to compute views for recently incoming data from various data sources. It is possible to enrich the master data by joining other related data sources. The second task is to store and update the recent incoming data in real-time because the older data is stored in the batch layer.

3) The Serving Layer
The serving layer is a specialized, distributed database system. It indexes the batch views so that they can be queried in a low-latency and ad-hoc manner. Then the serving layer joins the batch layer results and the speed layer computations on the data. So, it can provide real-time computation results over the entire data.

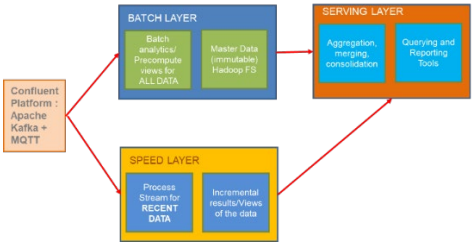

Fig. 6. The components of Lambda Architecture

III. PROPOSED APPROACH

A. Enterprise Layering & Lambda Architecture

The detailed component of *Lambda Architecture* used in this proposed framework architecture, as shown in Figure 7.

Sensors are Snort NIDS using subscribed community rules. The sensor can be placed anywhere over the Internet as long there is internet access to send the log to the DC. MQTT client is installed inside the sensor to send the alert data. The sensor uses MQTT Protocol to publish the alert data to the MQTT Server located on the defence centre.

MQTT Server, located in the DC, acting as MQTT message broker and equipped with a connector to distribute data from MQTT broker to Apache Kafka Broker that is on the same network as the defence centre. Bridging process, connecting between message inside MQTT Server and Apache Kafka.

Apache Kafka with multi-broker configuration is located in the same network as the defence centre. It streams the data to Apache Spark Instances.

Data storing jobs to HDFS, the data streamed from Apache Kafka will be stored inside the HDFS through Apache Spark. Stored data is raw immutable data. We can enrich the data with additional

attributes like local date-time, geographic information, and readable timestamp in this stage.

The data stored used inside the HDFS for batch processing using Apache Spark, that is, data aggregation in minutely, hourly, daily, weekly, monthly, and yearly. The results stored as batch processing in the form of JSON files in HDFS. Our main reason to reduce the network latency when batch processing or aggregation occurs.

Apache Spark used as structured streaming to perform automatic data aggregation sent by Apache Kafka. The source of data used in structured streaming is originated directly from Apache Kafka.

The result of data aggregation is stored in its respective aggregation table. There are two states of data results, temporary and permanent. Permanent data results will be stored inside the Cassandra table as time-series data. In contrast, the provisional data is published via Apache Kafka.

Both data results from batch and stream processing will be stored in the Apache Cassandra database. The result of all user queries and also the batch processing are stored in the Apache Cassandra.

Apache Cassandra, as the central database that acts as a permanent data storage and data source for all related services. Aggregated data is in the form of time-series data. The frequency of data retrieval and data write is relatively high, so to reduce the latency, Apache Cassandra is used due to its ability to serve the query using the clustered instance of the database.

Batch metric services is a collection of services that are responsible for doing a query from Apache Cassandra. Temporary data as a result of the stream processing will be published to Apache Kafka. Each aggregation result will be published on a different topic.

The temporary data will be published on a different topic. Apache Kafka is used as a message broker based on its function to provide real-time data. Due to the high write rate of Apache Kafka, it certainly makes the web services to consume the data at low latency and much more efficient needs of the resource.

By Apache Kafka, stream metrics will allow clients to get real-time data with low latency while still maintaining resource efficiency. The application of stream data can be in the form of a live graph that gets updated periodically.

Mongo DB is used to store the set of user information, i.e., user ID, organization ID, and sensor ID. Service for preserving Users and sensor management

Users can access all services through the REST API gateway.

ISSN 2636-9680
eISSN 2682-9266

REST API gateway is the only gateway between the client and the services. It also acts as the load balancer for each instance of service. Users can also access existing services through the provided gateway from their devices.

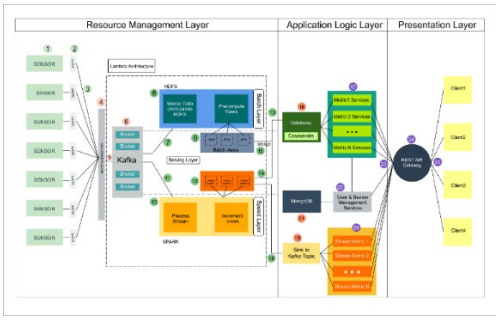

Fig. 7. Details of our proposed framework that implemented in Lambda Architecture

B. Metrics

Log data processing performed in Apache Spark consists of geographic information, timestamp, and time zone adjustments for sensors. These sensors may be placed in different locations and different time zones. We use Unix Epoch as the format of timestamp data sent by the sensor Aggregation is done in per second, per minute, per hour, per day, per month, and per year window time. The list of performed aggregations are: the event hit on company/device id, signature hit on company/device id, protocol hit on company/device id, protocol hit by source/destination port on company/device id, IP source/destination hit on company/device id, country source/destination hit on company/device id with each of the metrics counted by granularity as mentioned above.

IV. Experiment And Discussion

A. Sensor Deployment Time

The experiment is carried away to examine the time used to deploy the sensor that builds on top of Docker containers.

TABLE 1: Sensor deployment time.

	Debian Based Image	Alpine Based Image
Build Time	30 Minute	15 Minute
Image Size	623 MB	562 MB

From TABLE 1, we found that docker containers with Alpine as its base images are able to deploy the sensors two times faster than Debian-based containers.

B. Selecting Message Delivery

In order to find the best method for data transport between sensors with a defence centre, our experiment is as follows:

(1) The sensor generates 1000 messages/ second. Each message has a size of 924 bytes.
(2) The method that is compared is between MQTT and Apache Kafka via the internet network.

TABLE 2: Performance comparison of MQTT and Apache KAFKA

	Msg Rate: Latency MQTT	**Msg Rate: Latency With Kafka Pub**
Message (1000@924 Bytes)	210 msg/sec : 340ms	17 msg/sec : 31819 ms
Average Latency	543ms	33945ms

Our experiment shown in TABLE 2 found that MQTT has a higher rate and lower latency than Apache Kafka through the internet network.

C. Kafka Write Throughput

Data ingestion is the phase where data sent through the sensor to the data centre are collected in one place to be distributed to the next phase. Apache Kafka [16], [17] is one of the ingestion data frameworks that is reliable and has high latency when it comes to data ingestion. However, to be more convincing in selecting Kafka as data ingestion, the experiment will be carried out with the following specifications. The experiment will be carried out by loading the test on the single broker Kafka. Three experiments will be conducted, sending 50 million, 100 million, and 150 million data simultaneously. Then the average message/sec throughput will be calculated ".

TABLE 3: Load test on single Kafka Broker

Number of Messages (1 msg = 924 Bytes)	**Message / Second**	**Bytes / Second**
50.000.000	420.000	110 M
100.000.000	650.000	172 M
150.000.000	530.000	132 M

We send up to 100 million data, and brokers can still handle the requests. However, when the message reaches 150 million, it can be seen that there is a significant decrease in throughput rate, indicating that it takes additional new brokers to help handle concurrently. We summarize our load test in Table 3.

V. CONCLUSION

From our experiment can be inferred that there is a significant performance gap between MQTT and Apache Kafka when it comes to delivering data through the Internet. The limited bandwidth possibly causes this since data transportation takes place through the Internet. Meanwhile, Apache Kafka outperformed MQTT in data volume and ingestion speed at the local network where data ingestion mostly occurs.

ISSN 2636-9680
eISSN 2682-9266

Our proposed framework architecture successfully handles the computational load from the log data that originated from many sensors. The sensor built on top of Docker has a lower deployment time than the sensor's manual deployment. MQTT protocol also has better performance in sending the sensor to the defence centre than Apache Kafka due to its lightweight traits that lower latency than Apache Kafka through the Internet. Although Apache Kafka is excelled, MQTT can ingest many data in the local network needed to collect data from many sensors. We share the installation manual and source files of the Mata Elang at https://github.com/mata-elang-pens.

VI. REFERENCES

[1] "Snort - Network Intrusion Detection & Prevention System." [Online]. Available: https://www.snort.org/. [Accessed: 10-Jan-2019].

[2] "Mata Garuda." [Online]. Available: https://matagaruda.idsirtii.or.id/login. [Accessed: 21-Jan-2021].

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[4] Z. Chiba, N. Abghour, K. Moussaid, A. El Omri, and M. Rida, "A Cooperative and Hybrid Network Intrusion Detection Framework in Cloud Computing Based on Snort and Optimized Back Propagation Neural Network," *Procedia Comput. Sci.*, vol. 83, pp. 1200–1206, Jan. 2016.

[5] R. Zuech, T. M. Khoshgoftaar, and R. Wald, "Intrusion detection and Big Heterogeneous Data: a Survey," *J. Big Data*, vol. 2, p. 3, 2015.

[6] S. Irdoni, F. A. Saputra, and A. S. Ahsan, "Optimizing The Database of Mata Garuda by Using Partition Table," Politeknik Elektronika Negeri Surabaya, 2017.

[7] M. Hisyam, A. R. Barakbah, I. Syarif, and F. A. S, "Spatio Temporal with Scalable Automatic Bisecting-Kmeans for Network Security Analysis in Matagaruda Project," *Emit. Int. J. Eng. Technol.*, vol. 7, no. 1, pp. 83–104, Jun. 2019.

[8] M. Hisyam, F. A. Saputra, and J. Akhmad, "A Study of Implementing Data Mining Using Hadoop and Mahout for Mata Garuda Log Analysis," Politeknik Elektronika Negeri Surabaya, 2015.

[9] S. M. Othman, F. Mutaher Ba-Alwi, N. T. Alsohybe, and A. Y. Al-Hashida, "Intrusion detection model using machine learning algorithm on Big Data environment," *J. Big Data*, vol. 5, no. 34, 2018.

[10] G. P. Gupta and M. Kulariya, "A Framework for Fast and Efficient Cyber Security Network Intrusion Detection Using Apache Spark," in *Procedia Computer Science*, 2016, vol. 93, pp. 824–831.

[11] K. Peng, V. C. M. Leung, and Q. Huang, "Clustering Approach Based on Mini Batch Kmeans for Intrusion Detection

System over Big Data," *IEEE Access*, vol. 6, pp. 11897–11906, 2018.

[12] E. Viegas, A. Santin, A. Bessani, and N. Neves, "BigFlow: Real-time and reliable anomaly-based intrusion detection for high-speed networks," *Futur. Gener. Comput. Syst.*, vol. 93, pp. 473–485, 2019.

[13] "Get Started with Docker | Docker." [Online]. Available: https://www.docker.com/get-started. [Accessed: 12-Jan-2021].

[14] J. Xing *et al.*, "AsIDPS: Auto-Scaling Intrusion Detection and Prevention System for Cloud," in *2018 25th International Conference on Telecommunications, ICT 2018*, 2018, pp. 207–212.

[15] N. Marz and J. Warren, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*, 1st ed. Manning, 215AD.

[16] "Readings in information visualization: using vision to think." Morgan Kaufmann, 1999.

[17] F. Ullah and M. Ali Babar, "The Journal of Systems and Software Architectural Tactics for Big Data Cybersecurity Analytics Systems: A Review," *J. Syst. Softw.*, vol. 151, pp. 81–118, 2019.

ISSN 2636-9680
eISSN 2682-9266