

**Tugas** **Pendahuluan** **Proyek** **Akhir**

**DESAIN & IMPLEMENTASI BIG DATA PADA CLOUD-BASED IDS BERBASIS SNORT**

**DIMAS RIZKY HARSOYO PUTRO** **2110141011**

**D4** **TEKNIK** **INFORMATIKA** **DEPARTEMEN** **TEKNIK** **INFORMATIKA** **DAN** **KOMPUTER**

**POLITEKNIK** **ELEKTRONIKA** **NEGERI** **SURABAYA** **2017**

**A.** **JUDUL** **PROYEK** **AKHIR**

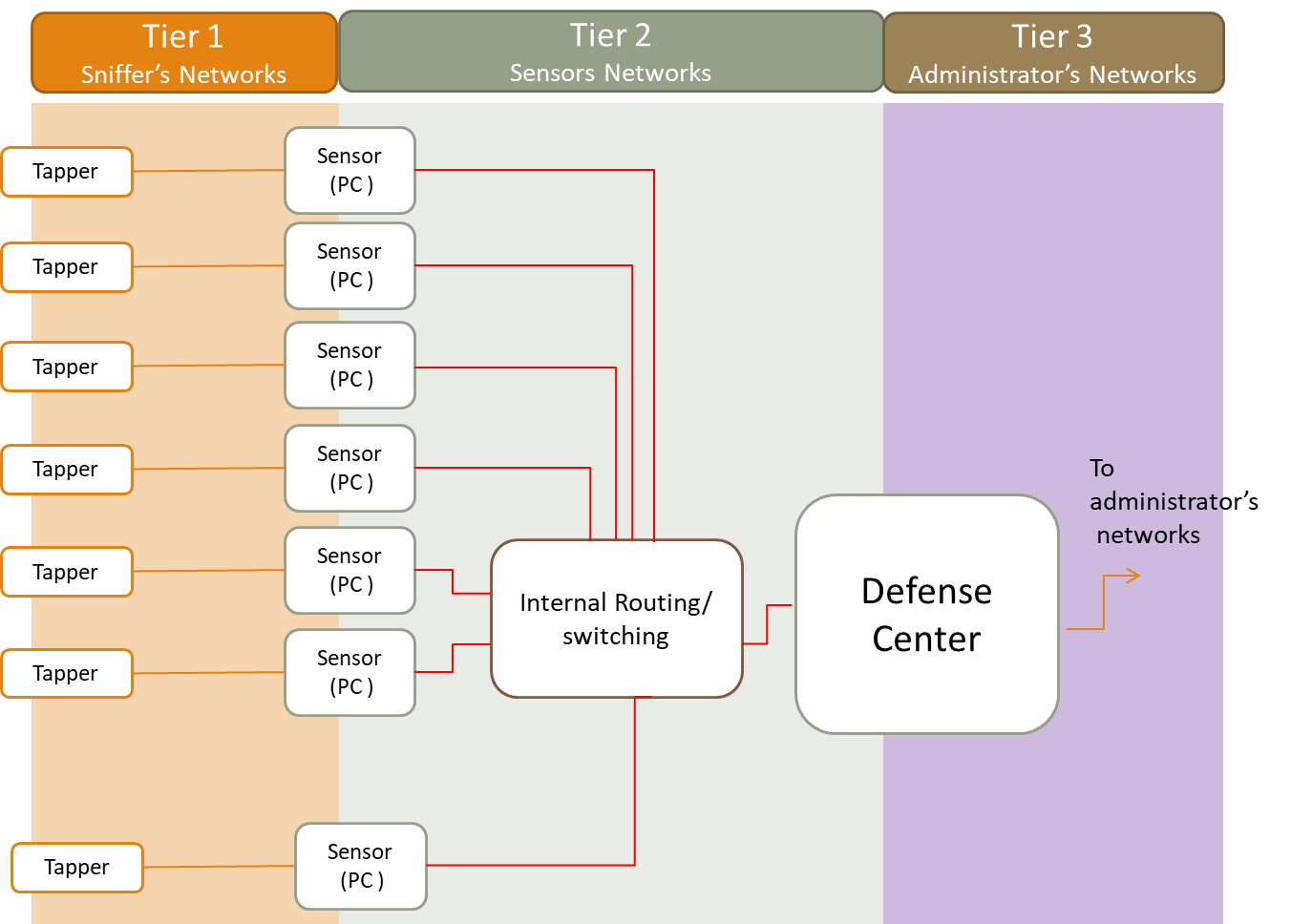
Desain & Implementasi Big Data pada Cloud-Based IDS Berbasis Snort

**B.** **PENDAHULUAN**

Cepatnya peningkatan volume data yang dihasilkan di internet oleh banyak kalangan mulai dari data pribadi, keuangan, hukum dan lainnya mengundang *hacker, cracker* atau *cyber-criminals*  yang lain untuk melakukan *cyber-crime*. Semakin meningkatnya jumlah serangan yang terjadi, dibutuhkan sistem yang dapat mencegah kegiatan ini untuk semakin berkembang. Solusi yang dibutuhkan juga semakin kompleks seiring dengan meningkatnya data secara heterogen dan semakin kompleks-nya infrastrukur jaringan yang dihadapi. Ditambah lagi dengan adanya *grid & cloud computing environment* yang bersifat terdistribusi, menjadikannya target favorit oleh para *intruders* untuk mencari kelemahan yang dapat dimanfaatkan untuk melakukan serangan [1].

Untuk mengatasi hal tersebut, intrusion-detection systems (IDSs) digunakan untuk menginvestigasi konfigurasi, *logs*, *network traffic*, dan *user action* untuk mengidentifikasi pola serangan yang ada [2]. Meskipun demikian, untuk mengatasi masalah yang sudah disebutkan di atas secara tuntas, IDS harus dapat berjalan di *grid & cloud computing environment*. IDS harus dapat mengamati tiap node, dan ketika serangan terjadi, dapat segera memberitahu node lainnya [1]. Untuk dapat memungkinkan adanya komunikasi antar node, diperlukan kompatibilitas antar host yang berbeda-beda, mekanisme komunikasi yang bermacam-macam, dan *permission control* terhadap sistem yang ada [3].

Mata Garuda merupakan *IDS (Intrusion Detection System)* yang memonitor lalu lintas data pada jaringan internet yang ada di Indonesia. Mata Garuda akan melaporkan suatu kejadian yang cirinya telah didefinisikan dalam sebuah rule. Kejadian-kejadian tersebut didapatkan melalui sensor yang terpasang di setiap *Network Access Point* yang ada di Indonesia. Sensor tersebut berfungsi untuk mengambil paket lalu meneruskannya ke *defense center*. Sensor yang digunakan oleh Mata Garuda adalah sensor berbasis *snort* yang bersifat private untuk setiap jaringan yang terpasang sensor. Mata Garuda menggunakan teknologi RDBMS untuk melakukan *storing* *data* serta pemrosesan data secara terpusat di *defense center*. Berikut adalah topologi Mata Garuda saat ini,



Gambar 1. Topologi monitoring Mata Garuda

Pada topologi Mata Garuda yang saat ini diimplementasikan, dilakukan penempatan *tapper* yang akan mengirimkan *packet* yang melewati jaringan ke sebuah sensor. Sensor berupa NIDS snort akan mendeteksi *packet-packet* tersebut apakah diantara *packet* tersebut ada yang sesuai dengan *ruleset* yang ada pada snort. Jika ditemukan *event* yang sesuai dengan *ruleset*, maka *event* akan diteruskan ke *defense center* untuk diproses, analisa dan dimasukan ke dalam database server. Setelah itu, data dapat ditampilkan dan diteruskan ke sistem milik administrator untuk dilakukan tindakan selanjutnya.

Pada kondisi yang sebenarnya, sensor-sensor yang dipasang tidak berada pada satu lokasi yang sama dengan *defense center* yang ada. Padahal dengan topologi yang ada, sensor diharuskan berada pada satu jaringan dengan *defense center*. Hal ini tentunya akan memakan *resource* yang cukup besar untuk mengintegrasikan sensor pada jaringan yang sama namun dengan lokasi fisik yang berbeda. Kondisi ini terjadi karena Mata Garuda belum mengadopsi *cloud computing environement* sehingga hanya dapat menerima *packet* dan *event* dari *private sensors.*

Dengan kondisi yang sudah dipaparkan di atas, tentunya Mata Garuda membutuhkan kapabilitas untuk dapat menangani pertumbuhan volume data yang semakin masif dan dapat berjalan di *cloud computing environment* agar sensor yang dapat digunakan persifat publik yang artinya tidak terikat penempatan fisiknya. Oleh karena itu, pengembangan framework & arsitektur sistem Mata Garuda menjadi sistem yang mengadopsi konsep Big Data dan dapat berjalan baik di *cloud computing environment* harus dilakukan agar dapat menyelesaikan permasalahan yang ada.

**C.** **PERUMUSAN** **MASALAH**

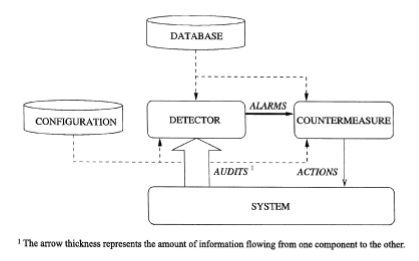
Mata Garuda saat ini menggunakan implementasi *RDBMS* dan penggunaan *data warehouse* untuk melakukan *storing data*. Hal tersebut mengakibatkan Mata Garuda tidak dapat mengadopsi konsep Big Data yang sangat dibutuhkan untuk memproses analisa data *event log* yang berukuran masif. Selain itu, penempatan fisik sensor mempengaruhi besarnya resource yang dibutuhkan karena sensor dituntut untuk berada pada satu jaringan dengan *defense center* dimanapun letak fisiknyadengan kata lain, sensornya yang dapat digunakan hanyalah *private sensor*. Tidak hanya itu, arsitektur Mata Garuda juga harus dapat berjalan baik di *cloud computing environment*.

**D.** **TINJAUAN** **PUSTAKA**

Tinjauan pustaka ini membahas tentang teori-teori penunjang dalam penyelesaian proyek akhir ini. Beberapa teori penunjang tersebut adalah :

*1.* *Intrusion* *Detection* *System*

*Intrusion Detection System* (IDS)dapat dideskripsikan secara sederhana yaitu detektor yang bertugas untuk memproses informasi dari sistem yang akan dilindungi [2]. Detektor ini memanfaatkan tiga macam informasi, yaitu *long-term* *information* terkait teknik yang digunakan utnuk mendeteksi intrusi, *configuration information* tentang keadaan terkini dari sebuah sistem dan *audit information* yang mendeskripsikan kejadian yang terjadi pada sebuah sistem. Tugas dari detektor adalah untuk menghilangkan informasi yang tidak penting pada *audit trail* dan menampilkan peta aksi yang dilakukan *user*. Setelah itu sebuah keputusan dibuat untuk menentukan kemungkinan bahwa aksi tersebut merupakan intrusi atau bukan.



Gambar 2 Intrusion-detection system sederhana

IDS setidaknya memiliki dua jenis, yaitu *host-based IDS* (HIDS) dan *network-based IDS* (NIDS). Pada masing masing jenis IDS, juga memiliki dua tipe pendeteksian yaitu, *misuse (rule-based)* dan *anomaly-based*. Masing-masing diukur berdasarkanakurasi, performa, *completeness, fault tolerance* dan *timeliness* [2]*.*

1. Akurasi

Ketidakakuratan muncul ketika IDS memberikan *flag postif intrusi* pada sebuah aksi yang padahal aksi tersebut bukanlah intrusi.

1. Performa

Performa dari IDS diukur menggunakan tingkat kecepatan sebuah IDS untuk melakukan audit sebuah kejadian. Jika performanya buruk, maka untuk mencapai *real-time* *detection* dapat dikatakan mustahil.

1. *Completeness*

*Completeness* berarti sebuah IDS tidak pernah gagal mendeteksi serangan, sebaliknya *incompleteness* terjadi jika sebuah IDS gagal mendeteksi sebuah serangan.

1. *Fault tolerance*

IDS sendiri harusnya kebal terhadap serangan yang ditujukan kepadanya. Hal ini dikarenakan IDS berjalan disebuah lingkungan OS komersil yang syarat akan celah keamanan

1. *Timeliness*

IDS harus dapat berjalan dan melakukan propagasi secepat mungkin untuk memungkinkan admin jaringan untuk bertindak secepat mungkin terhadap serangan yang dideteksi

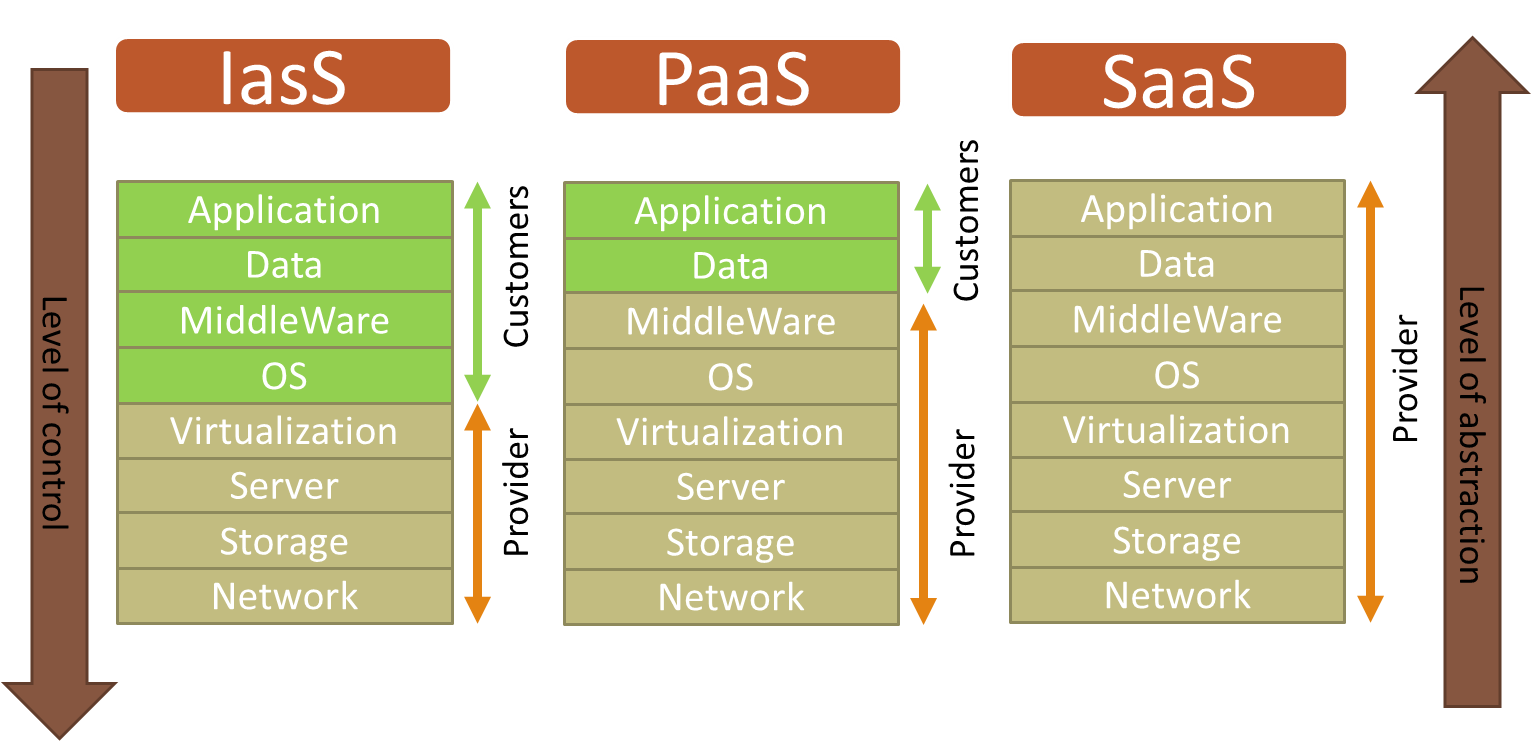
*Intrusion detection system* memenuhi dua topik *requirement*, yaitu *functional* dan *performance requirement* [4]*. Functional requirement* berarti (1) IDS harus dapat secara terus menerus memonitor dan melaporkan adanya intrusi, (2) IDS harus memiliki tingkat *false alarm* yang rendah, (3) IDS harus dapat menyediakan informasi yang cukup untuk memperbaiki sistem terkait intrusi yang terjadi, (4) IDS harus dapat mendeteksi dan bereaksi terhadap serangan yang terdistribusi, (5) IDS harus bersifat adaptif dengan topologi jaringan dan perubahan konfigurasinya. Namun, tentunya hal ini bergantung dengan apa tujuan dari IDS itu sendiri, dan fakanya banyak solusi IDS yang hanya fokus ke *alerting* kepada admin tanpa memberikan saran pencegahan

*Performance requirement* meliputi (1) intrusi harusnya dideteksi secara *real-time* sebab intrusi harus segera dilaporkan untuk meminimalisir adanya kerusakan, (2) IDS harus bersifat *scalable* agar dapat menagani beban komputasional dan komunikasi.

Terdapat beberapa tantangan yang dihadapi oleh IDS, yaitu berupa (1) tingginya tingkat *false positive*, (2) ketidak efisienan suatu IDS jika dihadapkan dengan *event* berjumlah besar, (3) rentan terhadap serangan.

2. Cloud Computing

*Cloud computing* adalah model komputasi yang seluruh sumber daya komputasinya tergabung menjadi satu, tersedia secara *on-demand*, sesuai permintaan, sesuai kebutuhan dan dapat tersedia dengan cepat serta pengoperasiannya mudah meskipun tanpa adanya campur tangan dari pihak provider. Terdapat tiga model umum yaitu, *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) dan *Software as a Service* (SaaS). Teknologi *Cloud computing* memungkinkan adanya (1) cakupan jaringan yang luas, (2) *powerful, inexpensive server computers*, (3) *high-performance virtualization for commodity hardware* [5]*.*



Gambar 3. Model layanan teknologi cloud computing

Model *cloud computing* menawarkan (1) adanya penghematan anggaran yang besar dan meningkatnya kecepatan dan ketangkasan dalam kapabilitas di bidang IT jika teknologinya diterapkan oleh konsumen, (2) lebih mudah, cepat dan efisien dalam penggunaan layanan, (3) tidak perlu mengeluarkan biaya tambahan untuk *maintenance* *server* ataupun *storage*, (4) tidak perlu mengeluarkan biaya tambahan untuk mengoperasikan *data center*, dan (5) mudahnya akses ke aplikasi dimanapun dan kapanpun.

Dalam sudut pandang *provider*, *cloud computing* (1) memudahkan mereka untuk meraih konsumen, (2) rendahnya biaya yang dibutuhkan untuk menyajikan layanan, dan (3) kemampuan untuk menekan biaya pengoperasian *data center* .*Cloud computing* sangat disarankan untuk diterapkan pada pemerintahan dan industri yang sedang mengalami kesulitan ekonomi.

Meskipun Begitu, terdapat beberapa tantangan yang harus dihadapi dalam pengimplementasian model ini, terdapat tujuh tantangan antara lain [6],

* *Privileged user access,* informasi yang ada akan dikirim dari *client* melalui Internet, tentunya hal ini menimbulkan resiko dan isu terkai *data ownership*. Konsumen harusnya mengetahui terlebih dahulu seluk beluk penyedia layanan jasa dan regulasi yang dimiliki sebelum masuk lebih dalam.
* *Regulatory compliance,* *client* adalah yang bertanggungjawab atas keamanan dari solusi yang mereka gunakan, karena *client* sendiri juga dapat memilih *provider* mana yang memiliki regulasi untuk mengijikan audit keamanan dan mana yang tidak.
* *Data location*, *client* mungkin saja tidak memiliki hak untuk mengetahui dimana letak data miliknya disimpan, hal ini bergantung dengan kontrak yang telah disetujui
* *Recovery,* setiap *provider* harusnya memiliki protokol *recovery* untuk setiap kejadian yang terjadi untuk melindungi data konsumen
* *Investigate support,* jika *client* mencurigai adanya aktivitas *provider* yang salah, *client* mungkin saja tidak memiliki kekuatan untuk melakukan investigasi lebih lanjut pada *provider*.
* *Long-term viability,* merujuk ke kemampuan untuk membatalkan kontrak dan menarik seluruh data yang ada jika *provider* dibeli oleh perusahaan lain.

4. Cloud Based IDS

*Cloud computing Intrusion Detection System* mengintegrasikan antara pengetahuan (*knowledge*) dan analisa perilaku (*behavior)* untuk mendeteksi adanya intrusi [6]. IDS dapat dikategorikan sebagai IaaS karena model ini yang paling fleksibel jika diterapkan pada IDS, IaaS memberikan konsumen lebih banyak opsi, ada empat opsi penempatan dan *depployment* dari IDS [7], antara lain,

*-* Ditempatkan di *Virtual Machine*, menempatkan *Intrusion Detection* pada VM memungkinkan *user* untuk memonitor aktivitas sistem dan mendeteksi serta memberikan *alert* jika ada isu yang terjadi

*-* Di *Hypervisor* atau *host system*, menempatkannya di posisi ini memungkinkan *user* untuk tidak hanya mengawasi *hypervisornya* melainkan dapat mengawasi seluruh *packet* yang melewati seluruh VM yang ada di *hypervisori* tersebut. Namun tentunya hal ini menimbulkan isu terhadap performa atau hilangnya informasi karena jumlah datanya terlalu besar yang disebabkan karena lokasi IDS nya tersentralisasi.

*-* Di *Virtual Network*, memungkinkan untuk mengawasi *virtual network* dan mengawasi *traffic* diantara VM dan *host*.

*-* Di *Traditional network,* memungkinkan untuk mengawasi *traffic* yang melewati infrastruktur *traditional network.*

Ada beberapa hal yang harus diperhatikan dalam melakukan *deploy* IDS di *environment cloud* [7] yaitu,

* *Provider* menempatkan ID (*Intrusion Detection*) / sensor di tempat tertentu yang nantinya mengirimkan data ke IDS milik *provider*.
* Konsumen harus memiliki *service-level agreement* (SLA) perihal tempat pemasangan sensor oleh *provider/*konsumen itu sendiri
* Jika yang memasang sensor adalah konsumen, maka sensor tersebut harus terintegrasi ke infrastruktur *monitoring* dan *alerting* milik *provider*.
* Untuk kedepannya, akan ada pihak ketiga yang dapat menyediakan IDS yang berjalan di cloud dan layanannya berupa *add-on* untuk melakukan pemasangan sensor di *environement* konsumen

Prinsip kerja dari *cloud-based intrusion detection system* adalah untuk [8] (1) mengurangi kebutuhan untuk melakukan training pada staff ahli baru, (2) mengurangi kebutuhan *licensing* suatu sfotware *security* baru, (3) tidak perlu membeli hardware baru, dan (4) dapat di *deploy* serta berjalan dimanapun seperti kaidah *cloud computing*, selain itu (5) konsumen juga mendapat kesempatan untuk mengatur dan mengadministrasi sendiri sistem IDS dan sensornya demi memberikan rasa nyaman dan aman pada konsumen.

3. Snort

Snort merupakan *Network-Based Intrusion Detection System* berlisensi *open source* yang dapat melakukan *packet logging* dan *traffic analysis* secara real-time. Dibuat oleh Martin Roesch pada tahun 1998. Snort merupakan *rule-based NIDS* yang artinya Snort menggunakan *rule* untuk mendeteksi adanya serangan pada jaringan. Snort akan melakukan tindakan yang sebelumnya telah ditentukan ketika mendeteksi adanya serangan yang sesuai dengan *ruleset* yang telah ditentukan.

4*.* HDFS

*Hadoop Distributed File System* adalah *distributed file system* berbasis *Java* yang digunakan untuk menyimpan file dalam jumlah yang masif dan disimpan secara terdistribusi di dalam banyak node komputer yang saling berhubungan. HDFS dibangun berdasarkan konsep *MapReduce* dan *Google File System* yang dikembangkan Google.

5. Apache Spark

Apache Spark adalah *engine cluster computing* open sourceyang khusus didesain untuk memproses big data. Dilengkapi dengan modul untuk melakukan *streaming,* SQL, *machine learning* dan *graph processing.* Apache Spark dapat berjalan berjalan di platform yang berbeda seperti Hadoop, Mesos, dan lainnya, serta dapat mengakses data dari sumber yang berbeda seperti HDFS, Cassandra, HBase, S3 ataupun sumber data yang lain. Spark memungkinkan untuk pembuatan aplikasi parallel dengan bahasa pemrograman yang berbeda-beda meliputi Java, Scala, Python, dan R.

Apache Spark dapat dikonfigurasikan agar berjalan sebagai *distributed system* menggunakan fitur RDDs (Resilient Distributed Datasets yang memungkinkan pemrosesan data dilakukan secara paralel dan terdistribusi [9].

Pada sisi kecepatan, Spark mengadopsi dan mengembangkan model *MapReduce* untuk mendukung tipe komputasional yang bermacam-macam secara lebih efisien. Spark menggunakan *multi-staged in-memory processing scheme* yang menghasilkan 100x lebih cepat waktu pemrosesan dibandingkan dengan pemrosesan menggunakan *MapReduce* [10].

6. Mata Garuda

Mata Garuda adalah IDS berbasis Snort IDS. Mata Garuda memantau lalu lintas jaringan internet di Indoneisa dan mengenali sebuah serangan atau kejadian lain yang sesuai dengan *ruleset* yang ada lalu menampilkannya dalam bentuk laporan yang dapat dengan mudah dibaca.

Prinsip dari Mata Garuda sendiri adalah *SLAD*, yaitu *SEE, LEARN, ADAPT & DECIDE.*

* See

*See* berarti Mata Garuda tidak hanya melihat serangan pada fase serangannya saja. Prinsip ini terbentuk karena pada teknologi monitoring yang klasik, yang dilihat hanyalah fase serangannya saja.

* Learn

*Learn* memadukan Mata Garuda dengan konsep *machine learning* untuk mempelajari *traffic* lokal yang sarat akan *customized development.* Terbentuk karena teknologi sebelumnya tidak fleksibel akan perubahan yang ada.

* Adapt

*Adapt,* pengembangan Mata Garuda dilakukan secara modular, sehingga memudahkan untuk dilakukan pengembangan kedepannya. Terbentuk karena teknologi monitoring klasik terpaku pada struktur *tools* yang sama.

* Decide

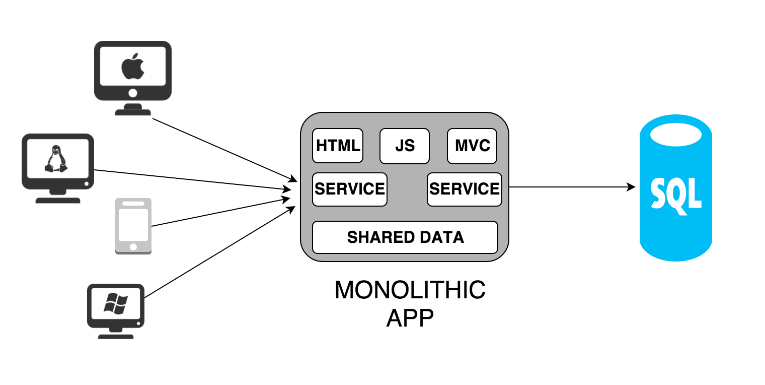
*Decide*, berarti Mata Garuda dapat mengurangi *false alarm* dengan penggunaan dan implementasi *intelligence technology*. Terbentuk karena pada teknologi sebelumnya banyak menghasilkan *false alarm*.

8. Microservices

*Microservices* adalah aplikasi sederhana yang dapat didistribusikan, dikembangkan dan dijalankan secara independen, serta memiliki tanggungjawab tunggal. Tanggungjawab tunggal disini berarti, hanya dapat melakukan satu tujuan tertentu namun mudah dipahami [11].

*Microservices* merupakan sebuah pendekatan arsitektur pada model pengembangan aplikasi, dimana sebuah aplikasi dibagi menjadi beberapa komponen kecil, yang setiap komponennya dikembangkan, dikemas dan didistribusikan secara individu [12].

Arsitektur ini merupakan pengembangan lanjut dari model arsitektur *monolithic*, dimana arsitektur *monolithic* dibangun sebagai satu kesatuan utuh yang berisi seluruh komponen yang dibutuhkan.

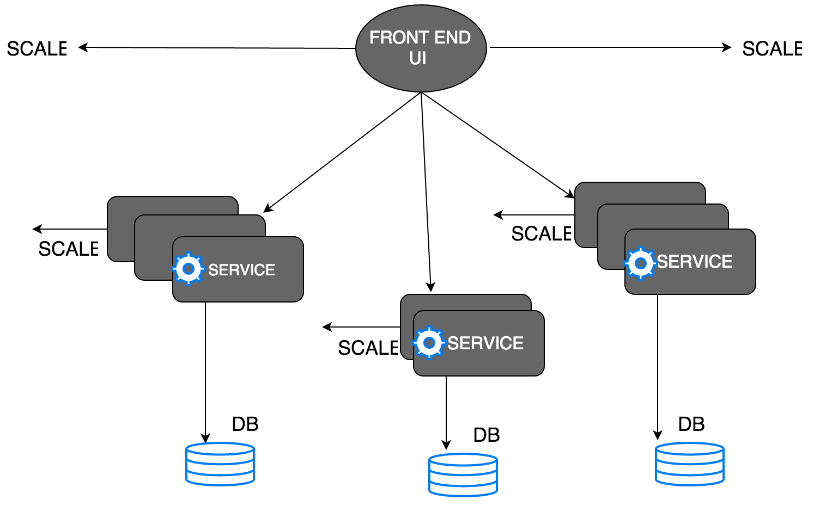


Gambar 4. Arsitektur Monolithic

Arsitektur *monolithic* memproses *user business logic* pada satu proses aplikasi. Keseluruhan fitur aplikasi ditulis dalam modul yang berbeda lalu akan di*pack* dalam satu kesatuan aplikasi. Pengembangan aplikasi *monolithic* tergolong sangat sederhana yang hanya cukup dengan membuat *multiple instance* baru dan menempatkannya tepat dibalik *load balancer* yang ada. Namun, ada beberapa tantangan dalam implementasi pengembangannya, antara lain [13] [12],

* Terkadang ukuran *package application* yang menerapkan arsitektur *monolithic* akan berkembang menjadi sangat besar seiring dengan adanya penambahan fitur. Pada saat aplikasi tumbuh menjadi aplikasi berukuran sangat besar, (1) pengembangan serta distribusi aplikasi tersebut akan sangat sulit dilakukan, (2) *bug tracking & fixing* akan menjadi pekerjaan yang melelahkan dan memakan *resource besar.*
* *Deployment* aplikasi kedepannya akan sangat *costly* untuk dilakukan. Hal tersebut disebabkan karena ketika beberapa komponen aplikasi dilakukan *bug fixing*, untuk melakukan *deploy* aplikasi harus dilakukan *re-deploy* aplikasi secara keseluruhan, padahal *bug fix* yang dilakukan hanyalah merupakan *minor bug fix* pada beberapa komponen.
* *Scaling* pada aplikasi *monolithic* juga akan sangat berat dilakukan. Untuk melakukan *scaling* harus dilakukan pada keseluruhan aplikasi, meskipun hanya butuh satu atau beberapa komponen tambahan untuk melakukan *scaling*.
* Susahnya mengimplementasikan teknologi baru atau bahasa pemrograman baru merupakan tantangan pada sebuah aplikasi *monolithic*.
* Jika satu saja komponen mengalami kegagalan atau *failure*, maka seluruh set komponen aplikasi juga akan ikut mengalami *failure.*

Tantangan yang telah dipaparkan di atas dapat dijawab dengan adanya pendekatan arsitektur *microservices.* Kelebihan utama dari *microservices* adalah (1) setiap komponen berbeda dapat di-*deploy* secara independen, tanpa harus melakukan *deploy* seluruh aplikasi hanya karena ada perubahan yang kecil.



Gambar . Arsitektur Microservices

Seperti yang ditunjukan pada *gambar 5*, komponen UI(*User Interface*) terpisah dari *core logic* dan dapat dilakukan *scaling* secara independen. Juga dapat diketahui bahwa komponen *backend* juga terpisah menjadi *service* yang independen. Kelebihan yang lain adalah (2) Setiap *service* dapat menggunakan tipe *database* yang berbeda-beda, (3) Setiap *service* cukup dapat di-*manage* oleh tim kecil hal ini juga memberikan keleluasaan untuk masing-masing tim untuk memilih teknologi apa yang sesuai untuk digunakan dalam pengembangan *service* yang di-*manage*, (4) Setiap *service* dapat di *scale* secara independen yang mana akan menjadikan prosesnya mudah dan tidak memakan terlalu banyak *resource*.

Namun, disamping kelebihan dari arsitektur *microservices* terdapat beberapa isu dan tantangan pada saat implementasinya, yaitu

* Pada arsitektur *monolithic* komunikasi antar komponen dilakukan menggunakan *internal procedure* atau cukup dengan memanggil sebuah *method*. Berbeda dengan arsitektur *microservices* dimana *developer* harus berurusan dengan *logic* untuk komunikasi antar komponen, juga harus berurusan dengan *error handling* diantara komunikasinya.
* Melakukan *testing* pada arsitektur *microservices* merupakan pekerjaan yang cukup memakan waktu dan tenaga dibandingkan dengan melakukan *testing* pada arsitektur *monolithic*, hal ini disebabkan karena adanya kemungkinan bahwa satu *service* memiliki hubungan dengan *service* yang lain.
* Harus adanya *service discovery mechanism*, yang fungsinya untuk membantu suatu *service* menemukan *service* lainnya

Pada *cloud environment*, sebuah aplikasi yang di *deploy* secara *cloud* sebagai PaaS(*Platform as a Service*), Iaas(*Infrastructure as a Service*), ataupun SaaS(*Software as a Service*) dituntut untuk memiliki kemampuan dan kapabilitas untuk (1) *auto scaling*, (2) *continuous delivery*, (3) *hot deployments*, (4) *high availability* dan (5) *dynamic monitoring* [14]*.* Pada awalnya, untuk memenuhi tuntutan tersebut, para *provider* memilih untuk mengadopsi pendekatan arsitektur *monolithic*. Namun, seperti yang sudah dipaparkan pada paragraf tentang tantangan yang harus dihadapi pada arsitektur *monolithic* di atas tentunya tidak cukup untuk memenuhi tuntutan aplikasi pada *cloud computing environment*. Dengan kondisi yang dihadapi, maka para *cloud service provider* segera beralih dari arsitektur *monolithic* menjadi arsitektur *microservices* yang kelebihannya sudah dipaparkan pada paragraf di atas.

7. Apache Kafka

Apache Kafka merupakan *message broker* untuk mengalirkan data dari suatu sumber data ke sistem target terdistribusi secara cepat dan efektif. Kafka didesain khusus agar berjalan baik di system terdistribusi yang memiliki tingkat *throughput* yang sangat tinggi. Dibandingkan dengan *message broker* yang lain, kafka memiliki *throughput* yang lebih baik, *built-in partitioning, replication* dan *fault-tolerance*.

Kafka memiliki tiga *key capabilities*, yaitu

* Memungkinkan untuk melakukan *publish & subscribe* pada record yang berjalan secara *stream*.
* Memungkinkan untuk menyimpan *stream of records* secara *fault-tolerant*
* Memungkinkan untuk memproses *stream of records* secara bersamaan pada saat *record* tersebut berjalan.

Kafka umum digunakan papda aplikasi yang memiliki karakteristik, (1) membutuhkan *real-time streaming* *data pipeline* yang handal untuk mengirim data antar system atau aplikasi, (2) *real-time streaming application* yang membutuhkan proses transform atau beraksi terhadap *stream of data* [15].

**E.** **PENELITIAN** **TERKAIT**

Berikut adalah penelitian yang pernah dilakukan dan relevan dengan proyek akhir ini.

1. oleh G. I. Wowiling, F.A. Saputra, I.U. Nadhori [16]

Penelitian ini memberikan solusi permasalahan terhadap masalah penyajian data *report* dari NIDS Snort menggunakan database Postgre berbentuk web apps modular untuk memudahkan pengembangan aplikasi. Permasalahan yang diangkat adalah masih terbatasnya kemampuan IDS snort dalam menampilkan laporan kejadian serangan, yaitu hanya berupa teks dan tentunya akan menyulitkan admin jaringan untuk menganalisa dan mengidentifikasi laporan kejadian serangan yang dialami pada jaringan yang diawasi. Selain itu, aplikasi yang sudah ada untuk menampilkan laporan event snort terbilang sulit untuk dikembangkan karena arsitekturnya yang tidak modular.

2. Optimasi Desain Database Mata Garuda dengan Menerapkan Partition Table oleh S. Irdoni, F.A. Saputra, dan A. S. Ahsan [17]

Permasalahan yang diangkat dalam penelitian ini adalah masalah terkait aplikasi Mata Garuda, dimana aplikasi tersebut menggunakan PostgreSQL 9.1 yang hanya mampu menangani data dalam jumlah 4-5 juta pada secara efisien per query, untuk jumlah data melebihi jumlah tersebut akan mengakibatkan query berjalan sangat lambat. Hal tersebut disebabkan karena data disimpan ke beberapa tabel yang semakin lama ukurannya semakin besar, lalu penggunaan query join digunakan untuk mengakses tabel dengan ukuran yang besar tersebut. Permasalahan lainnya adalah karena topologi yang digunakan masih menggunakan hanya satu server database OLTP, sehingga beban yang harus ditangani oleh server sangatlah besar. Penulis mengajukan solusi permasalahan berupa penggunaan desain partition table sebagai desain database untuk Mata Garuda. Hasilnya waktu komputasi menjadi jauh lebih cepat dan memperbaiki performa Mata Garuda dalam menangani data yang berukuran besar.

3. IDS Log Analisis Menggunakan Hadoop dan Mahout untuk Data Mining pada Mata Garuda oleh M. Hisyam, F.A. Saputra dan J. Akhmad [18]

Penelitian ini mencoba untuk melakukan pemrosesan snort *log file* dengan menggunakan prinsip *Big Data* di aplikasi Mata Garuda. Dengan sistem terdistribusi, metode *data mining* dilakukan terhadap data geolocation untuk mendapatkan lokasi serangan yang terjadi. Penulis menggunakan UDTF untuk melakukan *query* serta membandingkannya dengan *join query*. Sedangkan algoritma yang diterapkan dalam proses *mining* tersebut adalah K-means *clustering* untuk mendapatkan *cluster* dari GeoIP serangan. Hasilnya UDTF mampu mereduksi waktu komputasi menjadi 0.08 detik yang semula memakan waktu 3561 detik menggunakan *join query*. Pada penelitian ini juga dibuktikan bahwa penggunaan *distributed processing* untuk mengolah data Mata Garuda merupakan pilihan yang tepat karena data dapat diolah lebih cepat.

3. Real-Time Intrusion Detection System Using Multi-agent System oleh W. Laftah Al-Yaseen, Z. Ali Othman, dan M. Zakree Ahmad Nazri [19]

Penelitian ini mencoba untuk meningkatkan performa dari sistem *IDS* dengan cara mengurangi waktu pemrosesan data pada saat *IDS* melakukan analisa serangan pada lalu lintas data yang ada menggunakan *Multi-agent System*. Jumlah *agent* yang dilibatkan bersifat adaptif dan dapat berubah-ubah sesuai dengan ukuran lalu lintas data dan ketersediaan *resource* pada sistem, sehingga penggunaan *multi-agent* tidak membebani performa dari host. Hasil yang didapatkan menggunakan *Multi-agent System IDS* mampu mengurangi waktu pemrosesan analisa serangan sebesar 81% dibandingkan dengan *IDS* tradisional dengan tetap mempertahankan akurasi dari analisanya

3. Scalable Distributed Real-Time Clustering for Big Data Streams oleh Antonio Loureiro Severien [20]

Penelitian ini merupakan project pembuatan framework untuk penerapan algoritma *machine learning* pada data stream dengan volume yang sangat besar. Konsepnya adalah menggabungkan antara *machine learning, streaming model* dan *distributed computing* menjadi satu kesatuan. Hasil akhirnya berupa aplikasi bernama SAMOA (Scalable Advanced Massive Online Analysis) dengan fokusan hasilnya adalah pada aspek *scalability* dan *clustering quality*. Hasil penelitiannya menunjukan bahwa SAMOA dapat berjalan dengan baik pada *distributed environment* dan memiliki kapabilitas *scaling-up* untuk mengatasi *loads* yang lebih besar.

4. Real-time Hybrid Intrusion Detection System using Apache Storm oleh Goutam Mylvarapu, Johnson Thomas, Ashwin Kumar TK [21]

Penelitian ini mencoba menyelesaikan permasalahan lambatnya suatu IDS dalam menangani data dengan volume yang sangat besar. Solusi yang ditawarkan adalah dengan membangun *real-time hybrid IDS* dengan menggunakan Apache Storm. Apache Storm bertindak sebagai prosesor *real-time* untuk *big data stream-*nya, yang terdistribusi dan *fault tolerant*. Hybrid yang dimaksudkan adalah dengan adanya penggunaan dua algoritma *neural network* yaitu CC4 sebagai *anomaly-based detection* dan Multi Layer Perceptron sebagai *misuse-based detection*. Hasilnya menunjukan dengan *hybrid IDS* ini dapat berjalan di *distributed environment* dengan akurasi sebesar 89% dengan 4.32% tingkat *false positive*.

5. Performance Analysis of Network Intrusion Detection Schemes using Apache Spark oleh Manish Kulariya, PriyankaSaraf, Raushan Ranjan, Govind P. Gupta [22]

Penelitian ini melakukan pembandingan performa antara lima algoritma *machine learning* pada sebuah IDS dengan *distributed processing system* sebagai *engine* pemrosesan datanya. Apache spark digunakan sebagai *engine-*nya dan lima algoritma yang diimplementasikan antara lain adalah *logistic regression, support vector machines, random forest, gradient boosted decision* dan *naive bayes*. Komparasi dilakukan menggunakan dataset KDD99 sebagai variabel kontrolnya, dan hasilnya algoritma *random forest* yang memiliki performa terbaik yang dapat berjalan di *distributed environment* Apache spark.

**F.** **TUJUAN** **PROYEK** **AKHIR**

Tujuan dari proyek akhir ini adalah untuk membangun framework Mata Garuda dengan arsitektur yang mengadopsi konsep Big Data dan dapat berjalan menggunakan *public sensor* pada *cloud computing environment*.

**G.** **KONTRIBUSI** **PROYEK** **AKHIR**

Hasil dari proyek akhir ini dapat digunakan untuk meningkatkan kapabilitas Mata Garuda dalam menangani volume data yang masif dan dapat menerima data dari *public sensor* dalam *cloud computing environment*.

**H.** **METODE** **PROYEK** **AKHIR**

Untuk menyelesaikan proyek akhir ini langkah-langkah yang diambil ialah :

1. Studi Literatur

Studi literatur merupakan langkah awal dalam pengerjaan proyek akhir ini. Tahapan ini merupakan tahap yang penting untuk memelajari teori-teori serta konsep teknis maupun non-teknis yang menunjang dalam pengerjaan proyek akhir ini.

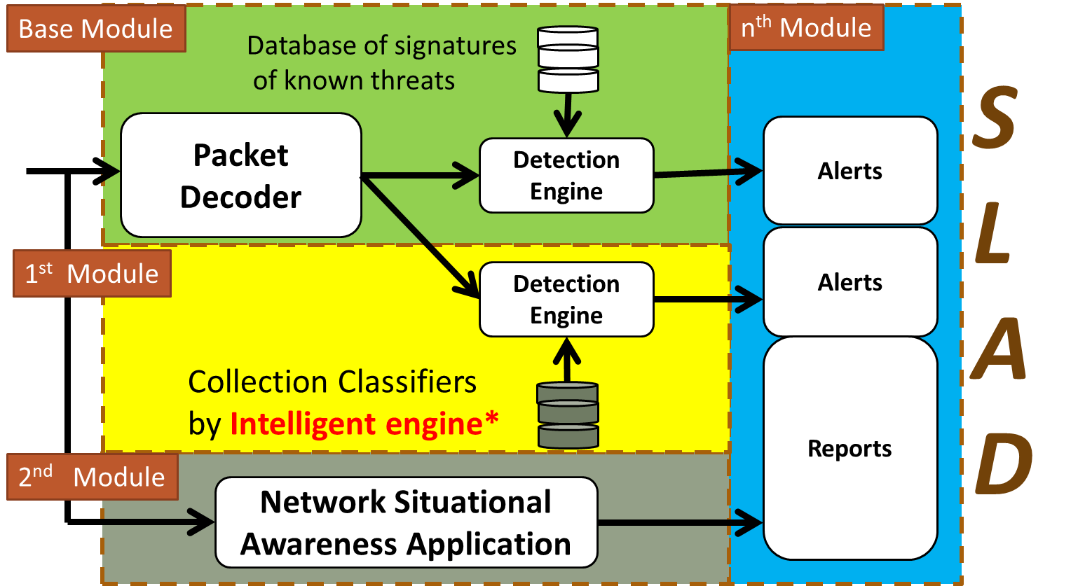
2. Pengumpulan data

Langkah selanjutnya dari penelitian ini adalah pengumpulan data log file dari Snort sebagai bahan analisa awal. Setelah data terkumpul, maka data tersebutakandiolahdenganteknik*textextraction*untukmembentukalgoritma dalam log parser.

3. Perancangan sistem

Setelah tahap studi literatur, langkah yang diambil selanjutnya adalah melakukan perancangan sistem.

Blok diagram sistem pada Mata Garuda yang sekarang adalah sebagai berikut,

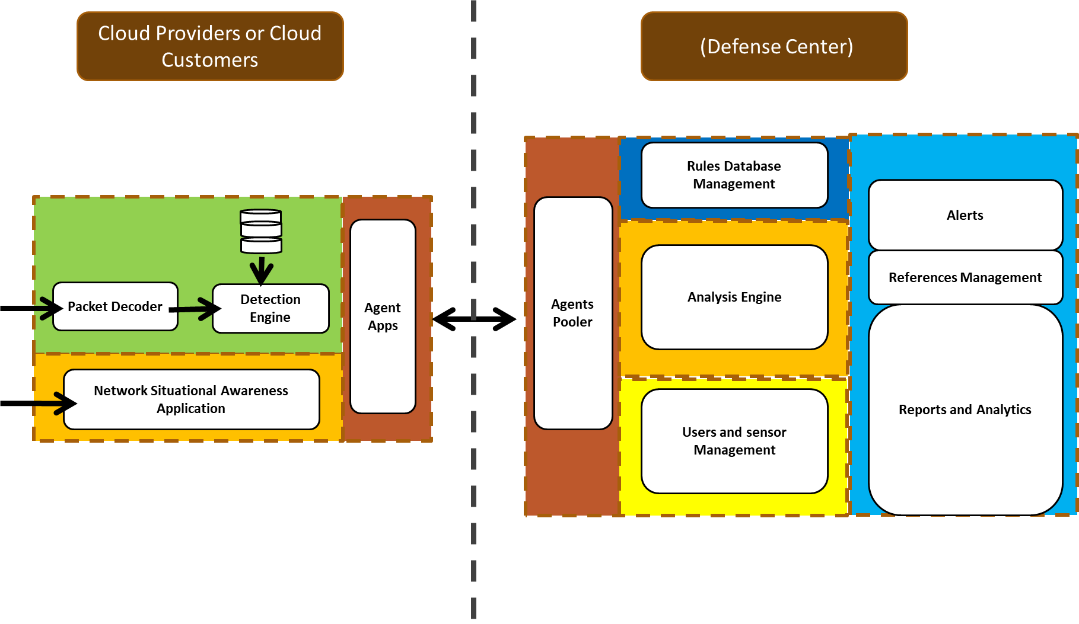


Gambar 6. Diagram sistem Mata Garuda yang sekarang

Pada diagram sistem tersebut, *base modul* merupakan modul penangkapan data *packet* dan pemeriksaan *packet* oleh sensor yang ada. Pemeriksaan didasarkan pada database *signature of known threats* menggunakan IDS snort sebagai *detection engine-*nya. Data *packet* dari *base module* juga akan dialirkan ke *1st module* untuk diklasifikasikan berdasarkan data *packet* yang sudah ada di database menggunakan *intelligent engine*, sehingga *base module* dan *1st module* akan menghasilkan *alert*.

*Data packet* tersebut nantinya akan diolah oleh *Network Situational Awareness Application* agar nantinya menghasilkan report dari *alert* dan *event* yang terjadi. Semua proses dilakukan dengan kondisi *base module* dan *1st* module berada pada satu arsitektur jaringan yang sama.

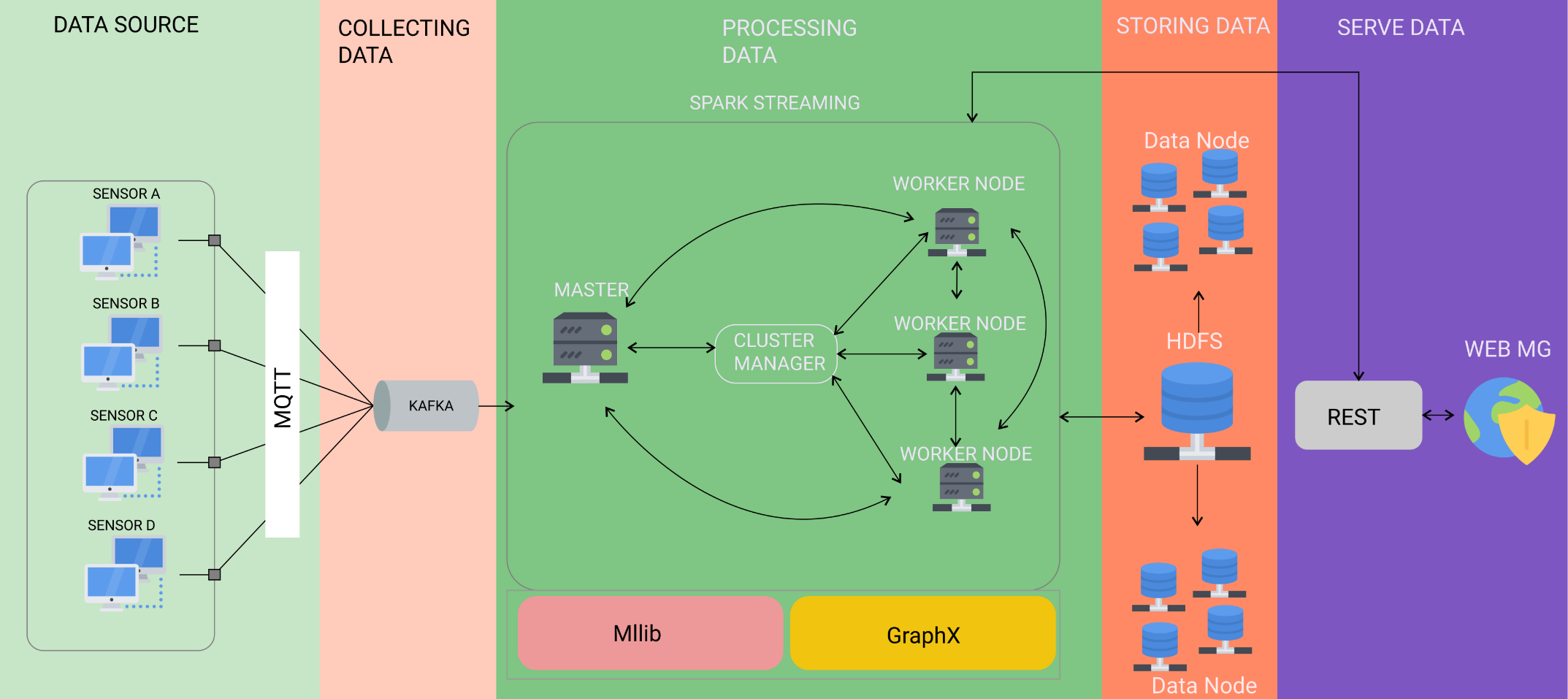
Sedangkan, untuk sistem yang akan dikembangkan, desain sistemnya lebih mengarah ke *cloud computing environment*. Secara umum, konfigurasi dari blok diagram sistem yang akan dibangun adalah sebagai berikut



Gambar 7. Diagram sistem cloud computing environment untuk Mata Garuda

Pada diagram sistem yang akan dibangun dirancang agar dapat berjalan di *cloud computing environment* dimana sensor atau *base module* dapat dipasang pada jaringan yang berbeda dengan *main module (defense center)*. Dengan ini, pada sisi pelanggan, atau penyedia sensor hanya bertindak sebagai penyuplai data *packet* atau *alert* yang nantinya akan diproses dan dianalisa pada *defense center*.

Untuk lebih spesifiknya, topologi *monitoring* yang diajukan sebagai pengembangan sistem terbaru dari Mata Garuda dapat dilihat pada gambar 4 di halaman selanjutnya.



Gambar 8. Topologi Sistem Keseluruhan

1. Sensor merupakan sensor berbasis snort yang menerima kejadian-kejadian yang ada di jaringan. Kejadian tersebut berupa *log file*. Untuk setiap kejadian yang terjadi akan dikirimkan *log file* ke *defense center* berupa *apache spark processing engine* lalu akan diproses dan dianalisa disana. Sensor dapat ditempatkan sebagai *public sensor* maupun *private sensor*. *Public sensor* dapat mengirimkan data *log file* melalui internet menggunakan protokol MQTT dan akan diteruskan ke *Apache Kafka Message Broker* untuk nantinya akan diteruskan ke *processing engine.*

Public Sensor

MQTT

Kafka Message Broker

Rule1

Rule2

Private Sensor

MQTT

Ruleset1

Ruleset2

MQTT Broker

MQTT Broker

Public Sensor

Private Sensor

Processing Engine

Gambar 9. Pengambilan data dari data source

1. Kafka

Kafka akan berperan sebagai message broker yang mengirimkan output dari sensor menuju ke *spark core engine* untuk nantinya diproses dan dimasukan kedalam *storage* HDFS.

Apache kafka yang digunakan adalah dengan model *publish* & *subscribe* dimana sumber data yang berasal dari sensor dikirimkan memiliki sebuah *topic*. Masing-masing sensor akan mengirimkan *log file* nya melewati MQTT broker dengan *topic* yang telah ditentukan. Kemudian MQTT broker milik sensor akan melakukan *publish* *log file* tersebut ke kafka producer dengan *identifier*-nya adalah *topic* itu sendiri.

Disisi lain, kafka akan melakukan *subscribe* terhadap *topic* yang di-*publish* oleh masing-masing sensor. Dengan cara itu sensor dapat mengirimkan *log file* secara *machine-to-machine* sehingga memungkinkan adanya *real-time* *queue transfer* antar MQTT broker milik sensor dan apache kafka.

Kafka akan melakukan *publish topic* ke apache spark untuk dilakukan *streaming process* dan analisa lebih lanjut.

Kafka Producer

Sumber Data (Sensor A, B, C)

Message Queue

TOPIC A

TOPIC B

TOPIC C

Spark Streaming

Gambar 10. Kafka messaging pipeline

1. Rancangan Aplikasi

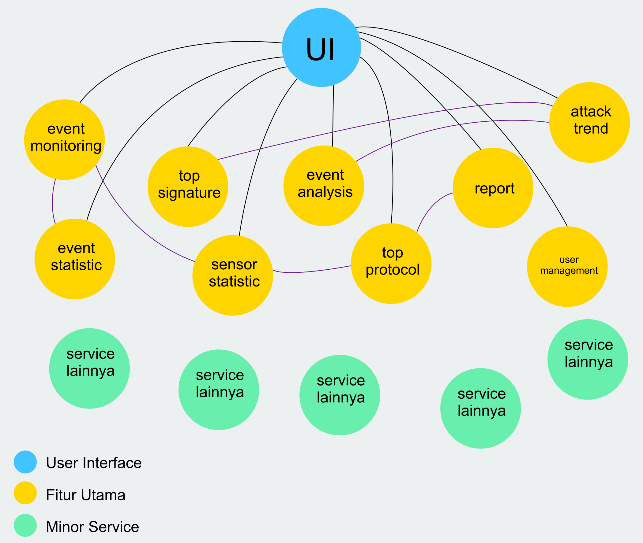
Aplikasi akan dibuat dengan pendekatan arsitektur *microservices* dimana untuk setiap komponen *service* merupakan komponen yang terpisah. Dengan implementasi *microservices* akan memudahkan pengembangan aplikasi untuk kedepannya dan juga memberikan kemudahan untuk melakukan *scaling* serta *maintenance*.

Selain *service* yang dibuat, juga akan dibangun web apps default dimana web apps tersebut dibangung berdasarkan *services* yang telah dibuat sebelumnya. Aplikasi yang dihasilkan akan berupa web apps dengan fitur dasar pada sebuah framework meliputi,

* + - Login
    - Admin Dashboard (CRUD) & Dashboard User
    - Mengatur HDFS
    - Konfigurasi
    - Membuat API key untuk sensor
    - Melihat Profile
    - Mengakses Report & Metric yang ada
    - Authentication untuk sensor baru
    - Read data melalui data source

1. REST API

REST API merupakan service yang digunakan untuk melakukan pengambilan data dan proses manajemen yang lainnya melalui *core engine* apache spark untuk ditampilkan ke web interface.



Gambar . Microservices

* + - Metric Mata Garuda

Dalam service-nya digunakan metric mata garuda antara lain adalah, *event monitoring, event statistics, top signature, top protocols, sensor statistics*, *daily-monthly-annually report, attack trend, event analysis, user-role-profile-menu management*

1. Web Interface

Web interface akan digunakan untuk menampilkan hasil query yang dilakukan oleh REST API. Laporan yang ditampilkan adalah berupa laporan terkait serangan dan kejadian yang ditangkap oleh sensor.

1. Apache Spark

Spark bekerja sebagai execution engine untuk pemrosesan data yang berasal dari sensor. Spark diintegrasikan dengan menggunakan SQL Hive. Spark bertugas untuk melakukan eksekusi query yang dilakukan oleh REST API

* 1. Streaming SQL

Dapat memungkinkan spark memproses data secara stream, sehingga memungkinkan untuk melakukan *write job* secara streaming

* 1. Mllib

Mllib merupakan library machine learning dari spark, bertugas untuk melakukan klustering data yang ada pada storage HDFS

* 1. GraphX

Merupakan library spark untuk melakuka komputasi graph-paralel

4. Implementasi

Dalam tahap ini akan dibangun arsitektur dan desain untuk mengaplikasikan konsep Big Data. Pengaplikasian dimulai dari mengkonfigurasikan bentuk output dari sensor Snort dengan menggunakan *Snort MQTT* agar dapat sesuai dengan sensor sensor yang lain, lalu mengkonfigurasikan dan mengintegrasikan apache kafka sebagai *pub-sub* dengan apache spark core engine agar dapat mengirimkan dan menerima pesan dari/ke spark engine.

Setelah itu akan dilakukan instalasi mesin master & slave node apache spark engine yang digunakan untuk melakukan *stream processing* terhadap data yang masuk ke engine, serta membangun konfigurasi *file system* menggunakan HDFS.

Tahap terakhir adalah membuat frontend web REST API untuk masing-masing metric yang ada. Melakukan debugging dan peningkatan performa pada program web

5. Pengujian dan analisa

Pada tahap ini, dilakukan pengujian terhadap implementasi desain sistem dengan menggunakan dataset skala lab yang sesuai dengan desain intrusi yang ada. Dan melakukan komparasi performa dengan desain sistem yang ada saat ini. Selain itu juga akan dilakukan testing dengan dataset dengan volume besar dan pemasangan sensor publik untuk melakukan uji kehandalan di *cloud computing environment*.

Selain itu juga akan dilakukan analytic serangan yang sedang mengalami *trending*, menurut SANS *Institute* *RSA 2017* *keynote session,* yaitu analytic serangan (1) *Ransomware* (SMB *malicious activities*), (2) *IoT (Internet of Thing) device attack,* (3) *Ransomware* pada *IoT* device, (4) *ICS (Industrial Controls Systems)* *Attack* yang memutus sumber listrik dan melakukan *disable* terhadap protocol *recovery* dan *smooth operations,* (5) *Attack* terhadap NoSQL Database

**I.** **JADWAL** **PELAKSANAAN**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Kegiatan | Bulan | | | | | | | | | | | | |
| 8 | 9 | 10 | 11 | 12 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Proposal ProyekAkhir |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Pengambilan Data |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |
| Implementasi/Coding |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |
| Debugging |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Testing &Analisa |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Penyusunan Buku PA |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |
| Sidang PA |  |  |  |  |  |  |  |  |  |  |  |  |  |

**J.** **PERSONALIA** **PROYEK** **AKHIR**

Mahasiswa

Nama

NRP Jurusan Agama

Jenis kelamin

Dosen Pembimbing 1 Nama

NIP

: Dimas Rizky H.P.

: 2110141011

: Diploma IV - Teknik Informatika : Islam

: Laki-laki

: Ferry Astika Saputra, ST, M.Sc

: 197708232001121002

Departemen

Bidang keahlian

Dosen pembimbing 2

Nama NIP

Departemen

Bidang keahlian

: Teknik Informatika

: Computer Network, Network Security

: Jauari Akhmad S.ST

: 2000000052

: Teknik Informatika

: Rekayasa Perangkat Lunak

**K.** **PERKIRAAN** **BIAYA** **PROYEK** **AKHIR**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Uraian | Jumlah | Harga Satuan(Rp) | Total (Rp) |
| 1 | RAM 8GB | 4 buah | 800.000 | 3.200.000 |
| 2 | Kertas A4 HVS 80gr | 1 rim | 40.000 | 40.000 |
| 3 | Kertas A5 | 1 rim | 40.000 | 40.000 |
| 4 | Tinta Printer | 4 buah | 25.000 | 100.000 |
| 5 | Internet | 8 bulan | 100.000 | 800.000 |
| 6 | Dokumentasi Laporan | 6 bundle | 25.000 | 150.000 |
| **TOTAL** **PENGELUARAN** | | | | **4.330.000** |

**L.** **DAFTAR** **PUSTAKA**

|  |  |
| --- | --- |
| [1] | K. Vieira, A. Schulter, C. B. Westphall and C. M. Westphall, "Intrusion Detection for Grid and Cloud Computing," *IT Professional,* vol. 12, no. 4, pp. 38-43, 2010. |
| [2] | H. Debar, M. Dacier and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks,* vol. 31, no. 8, pp. 805-822, 1999. |
| [3] | I. Foster, C. Kesselman, G. Tsudik and S. Tuecke, "A Security Architecture for Computational Grids," in *CCS '98 Proceedings of the 5th ACM conference on Computer and communications*, San Francisco, 1998. |
| [4] | W. Jansen, P. Mell, T. Karygiannis and D. Marks, "Applying Mobile Agents to Intrusion Detection and Response," NIST Interim Report, Gaithersburg, 1999. |
| [5] | (NIST) National Institute of Standards and Technology, "NIST Cloud Computing Program - NCCP," NIST, 13 Juli 2017. [Online]. Available: https://www.nist.gov/programs-projects/nist-cloud-computing-program-nccp. [Accessed 1 Agustus 2017]. |
| [6] | S. R. M, V. N and T. S, "Effective Analysis of Cloud Based Intrusion Detection System," *International Journal of Computer Applications & Information Technology,* vol. I, no. 2, pp. 16-22, 2012. |
| [7] | P. Cox, "Intrusion detection in a cloud computing environment," TechTarget, March 2010. [Online]. Available: http://searchcloudcomputing.techtarget.com/tip/Intrusion-detection-in-a-cloud-computing-environment. [Accessed 1 August 2017]. |
| [8] | W. Yassin, N. I. Udzir, Z. Muda, A. Abdullah and M. T. Abdullah, "A Cloud-Based Intrusion Detection Service Framework," in *2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, Kuala Lumpur, 2012. |
| [9] | H. Karau, A. Konwinski, P. Wendell and M. Zaharia, Learning Spark Lightning-Fast Data Analysis, Sebastopol: O'Reilly Media, 2015. |
| [10] | Apache Spark, "Apache Spark™ - Lightning-Fast Cluster Computing," Apache, [Online]. Available: http://spark.apache.org/. [Accessed 1 August 2017]. |
| [11] | A. Sill, "The Design and Architecture of Microservices," *IEEE Cloud Computing,* vol. III, no. 5, pp. 76-80, 2016. |
| [12] | S. Pillai, "Difference Between Monolithic and Microservices based Architecture," Slashroot.in, 28 12 2015. [Online]. Available: http://www.slashroot.in/difference-between-monolithic-and-microservices-based-architecture. [Accessed 2 August 2017]. |
| [13] | Mulesoft, "Microservices vs Monolithic Architecture | MuleSoft," Mulesoft, [Online]. Available: https://www.mulesoft.com/resources/api/microservices-vs-monolithic. [Accessed 2 August 2017]. |
| [14] | M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas and S. Gil, "Evaluating Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud," in *10th Computing Colombian Conference (10CCC)*, Bogota, 2015. |
| [15] | Apache Kafka, "Apachhe Kafka," Apache, [Online]. Available: https://kafka.apache.org/intro. [Accessed 1 August 2017]. |
| [16] | G. I. Wowilling, F. A. Saputra and I. U. Nadhori, TA MG Mas Gerry (?????), Surabaya: Politeknik Elektronika Negeri Surabaya, 2014. |
| [17] | S. Irdoni, F. A. Saputra and A. S. Ahsan, Optimasi Desain Database Mata Garuda dengan Menerapkan Partition Table, Surabaya: Politeknik Elektronika Negeri Surabaya, 2017. |
| [18] | M. Hisyam, F. A. Saputra and J. A. Nur Hasyim, IDS Log Analisis Menggunakan Hadoop dan Mahout untuk Data Mining Pada Mata Garuda, Surabaya: Politeknik Elektronika Negeri Surabaya, 2015. |
| [19] | W. L. Al-Yaseen, Z. A. Othman and M. Z. Ahmad Nazri, "Real-Time Intrusion Detection System Using Multi-agent System," *IAENG International Journal of Computer Science,* pp. 1-11, 2016. |
| [20] | A. L. Severien, "Scalable Distributed Real-Time Clustering for Big Data Streams," p. 2013, 25 June 2013. |
| [21] | G. Mylvarapu, J. Thomas and A. K. TK, "Real-time Hybrid Intrusion Detection System using Apache Storm," in *IEEE 17th International Conference on High Performance Computing and Communications*, New York, 2015. |
| [22] | M. Kulariya, R. Ranjan and P. G. Gupta, "Performance Analysis of Network Intrusion Detection Schemes using Apache Spark," in *International Conference on Communication and Signal Processing*, India, 2016. |
| [23] | E. N. Saad, K. E. Mahdi and M. Zbakh, "Cloud Computing Architectures Based IDS," in *IEEE International Conference on Complex Systems (ICCS)*, Agadir, 2012. |