



Arsitektur Microservice untuk Resiliensi Sistem Informasi

Hatma Suryotrisongko*

Jurusan Sistem Informasi, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember

Abstract

Resilient means be able to adapt to changes and errors/damage to the systems/infrastructure. In the study of information systems, resilience is often seen as nonfunctional requirements that can be seen from the element of scalability, reliability, maintainability and availability. This study continue the previous studies, namely the development of Open Source software for the management of associations / memberships. The software has been used in AISINDO member management system (Professional Association of Information Systems Indonesia). Continuation of the research study in the form of modeling and manufacturing of proof of concept of a distributed software architecture modifications, based MICROSERVICE and Docker-container, for Resilient Information Systems. The software research last year was refactored (rewriting program code), using a new software architecture models generated in this study. The test results on a system that has been made, indicating that the system with the proposed architecture has shown improvement in the quality aspects of resilience, eg when multiple nodes serve impaired, the system can still function properly. The contribution of this study was: MICROSERVICE implementation of management systems case study associations / memberships and, and then evaluate the model after the completion of construction project software refactoring into a system based microservices.

Keywords: Microservice Architecture, Docker, Software Refactoring, Software Quality, Resiliency

Abstrak

Resilien berarti tahan/mampu beradaptasi terhadap perubahan maupun kesalahan/kerusakan infrastruktur. Pada penelitian sistem informasi, resilience sering dilihat sebagai *nonfunctional requirements* yang dapat dilihat dari unsur *scalability*, *reliability*, *maintainability* dan *availability*. Penelitian ini melanjutkan hasil penelitian sebelumnya, yaitu pengembangan *software Open Source* untuk manajemen asosiasi/keanggotaan, dengan tujuan untuk meningkatkan kualitas *software* pada aspek resiliensi nya. *Software* tersebut saat ini telah dipakai pada sistem manajemen anggota AISINDO (Asosiasi Profesi Sistem Informasi Indonesia). Lanjutan dari penelitian-penelitian tersebut berupa penyusunan model dan pembuatan *proof of concept* dari modifikasi arsitektur *software* yang terdistribusi, berbasis *microservice* dan *Docker-container*, untuk *Resilient Information Systems*. Dilakukan *refactoring* (penulisan ulang kode program) dari *software* hasil penelitian sebelumnya, dengan menggunakan model arsitektur *software* baru yang dihasilkan di penelitian ini. Hasil pengujian pada sistem yang telah dibuat, menunjukkan bahwa sistem dengan arsitektur yang diusulkan telah menunjukkan peningkatan kualitas pada aspek resiliensi, misalkan ketika beberapa *node* *serve* mengalami gangguan, sistem dapat tetap berjalan sebagaimana mestinya. Kontribusi penelitian ini yaitu implementasi *microservice* untuk studi kasus sistem manajemen asosiasi/keanggotaan, lalu mengevaluasi model tersebut setelah selesai pengerjaan *refactoring software* menjadi sistem yang berbasis *microservices*.

Kata kunci: Microservice Arsitektur, Docker, Software Refactoring, Kualitas Software, Resiliensi

© 2017 Jurnal SISFO.

Histori Artikel : Disubmit 2 Desember 2017; Diterima 9 Januari 2017; Tersedia online 26 Januari 2017

*Corresponding Author

Email address: suryotrisongko@gmail.com (Hatma Suryotrisongko)

1. Pendahuluan

Di masa lalu, menjadi responsif dalam kasus gangguan yang tidak terencana sangat menyulitkan manajemen. Untuk IT, hal tersebut bahkan lebih menantang: Sistem yang dikembangkan dirancang hanya untuk memenuhi sifat yang telah ditetapkan, dan hanya menawarkan satu set fungsionalitas terprogram untuk penanganan eksepsi. *Resilience* meliputi reaksi atas gangguan di luar lingkup yang sudah dikenal sebelumnya. *Software*/sistem informasi dianggap *resilient* jika memiliki kemampuan menyesuaikan kebutuhan baru yang belum eksplisit ada di dalam desain IT sebelumnya [1]. Penelitian mengenai *resilience* pada bidang sistem informasi telah banyak dibicarakan semenjak tahun 2010, seperti pada *Enterprise Information Systems journal* Volume 4, Issue 2, 2010 yang membahas topik khusus mengenai *Resilient Enterprise Information Systems*, yaitu bagaimana sistem informasi yang digunakan pada organisasi besar (*enterprise*) seperti ERP, SCM, CRM, dll dapat dirancang untuk cukup *resilient* [2]. Pada salah satu paper di jurnal itu, dibahas beberapa desain arsitektur *software* & sistem informasi yang mampu meningkatkan performa *resiliency* pada sistem IT di organisasi [3].

Penelitian dan teknologi terus semakin cepat perkembangannya. Akhir-akhir ini, muncul tren baru dikalangan peneliti/praktisi *Software Architect*, yaitu *microservices*, dimana *software*/sistem informasi dirancang untuk terdistribusi dan memberikan layanan spesifik dan terfokus. Walaupun istilah *microservices* telah mulai diperkenalkan sejak tahun 2011 [4], namun pola desain arsitektur *software*/sistem informasi ini menjadi semakin banyak diteliti dan digunakan industri setelah munculnya teknologi *container-based virtualization*, yaitu *Docker* di akhir tahun 2014 [5]. Penelitian ini melanjutkan hasil penelitian sebelumnya, yaitu pengembangan *software Open Source* untuk manajemen asosiasi/keanggotaan, dengan tujuan untuk meningkatkan kualitas *software* pada aspek resiliensi-nya. *Software* tersebut saat ini telah dipakai pada sistem manajemen anggota AISINDO (Asosiasi Profesi Sistem Informasi Indonesia) dan juga telah dipublikasikan di *Wordpress Plugin repository* sehingga masyarakat internasional bisa turut menggunakan kode program ini. Setelah *software* tersebut diimplementasikan, hasil evaluasi menunjukkan kurangnya kapasitas sistem yang sudah ada dalam melayani jumlah member yang terus meningkat, sehingga perlunya peningkatan kinerja sistem dengan mencoba menerapkan pendekatan baru.

Pada paper ini, dipaparkan hasil penelitian berupa penyusunan model dan pembuatan *proof of concept* dari modifikasi arsitektur *software* yang terdistribusi, berbasis *microservice* dan *Docker-container*, untuk *Resilient Information Systems*. Kode program yang baru tersebut diimplementasikan pada AISINDO (Asosiasi Profesi Sistem Informasi Indonesia) sehingga bisa mendapatkan data empiris untuk evaluasi dan perbaikan model di penelitian berikutnya.

2. Tinjauan Pustaka/Penelitian Sebelumnya

2.1 Resilient Information Systems

Resilience engineering adalah salah satu topik penelitian multi disipliner yang melibatkan banyak peneliti dari disiplin ilmu *computer science*, *mechanical engineering*, *industrial engineering*, dan lain-lain. *Literature review* terkini tentang topik ini bisa dilihat pada [6]. Perbandingan definisi *resilience* dapat dilihat di Tabel 1.

2.2 Microservices

Ada banyak solusi yang diusulkan oleh para peneliti untuk dapat meningkatkan kualitas *software*/sistem informasi dari sisi *nonfunctional requirements* (*scalability*, *reliability*, *maintainability* dan *availability*). Mulai dari penggunaan teknik *mirroring*/*DRC*/*cloud*/sampai dengan pendekatan algoritma untuk dapat mengembalikan performa sistem setelah terjadi serangan [7].

Tabel 1. Definisi *resilience* dari perspektif berbagai disiplin bidang ilmu

Konteks	Definisi
<i>Physical systems</i>	Kecepatan di mana sistem kembali ke tingkat keseimbangan setelah perpindahan, terlepas dari osilasi menunjukkan elastisitas (ketahanan) [8]
<i>Ecological systems</i>	Ukuran kegigihan sistem dan kemampuan untuk menyerap perubahan dan gangguan dan tetap menjaga hubungan yang sama antara variabel [9]
<i>Ecological systems</i>	Kapasitas sistem untuk menyerap gangguan dan mereorganisasi saat menjalani perubahan sementara tetap mempertahankan fungsi yang sama, struktur, identitas dan umpan balik [10]
<i>Ecological systems</i>	Besarnya gangguan sistem dapat menyerap sebelum strukturnya didefinisikan ulang dengan mengubah variabel dan proses yang mengendalikan perilaku [11]
<i>Ecological systems</i>	Kecepatan di mana sistem kembali ke titik ekuilibrium setelah terjadi gangguan [12]
<i>Socio-ecological systems</i>	Kemampuan untuk mempertahankan fungsi sistem ketika terganggu atau kemampuan untuk mempertahankan elemen yang dibutuhkan untuk memperbaharui atau mereorganisasi jika gangguan mengubah struktur fungsi sistem [10]
<i>Socio-ecological systems</i>	Besarnya gangguan yang sistem dapat mentolerir sebelum transisi menjadi kondisi yang berbeda yang dikendalikan oleh satu set yang berbeda dari proses [13]
<i>Psychology</i>	Kapasitas untuk pulih dari keterpurukan [14]
<i>Disaster management</i>	Kemampuan unit sosial untuk mengurangi bahaya, mengandung efek bencana ketika mereka terjadi dan melaksanakan kegiatan pemulihan yang meminimalkan gangguan sosial dan mengurangi dampak dari gempa bumi di masa depan [15]
<i>Disaster management</i>	Ketahanan menggambarkan proses aktif meluruskan diri, belajar dan pertumbuhan. Konsep ini berkaitan dengan kemampuan untuk berfungsi pada tingkat psikologis yang lebih tinggi dan pengalaman sebelumnya [16]
<i>Individual</i>	Individu tangguh memiliki tiga karakteristik umum. Ini termasuk penerimaan realitas, keyakinan yang kuat bahwa hidup adalah bermakna dan kemampuan untuk berimprovisasi [17]
<i>Organisational</i>	Ketahanan mengacu pada kapasitas untuk rekonstruksi terus menerus [18]
<i>Organisational</i>	Ketahanan adalah kualitas mendasar untuk merespon secara produktif untuk perubahan signifikan yang mengganggu pola yang diharapkan dari acara tanpa memperkenalkan jangka perilaku regresif [19]

Microservice yang belum lama ini mulai populer di kalangan praktisi *Software Engineering* menawarkan pendekatan yang sedikit berbeda. Sistem Informasi enterprise yang pada umumnya dibangun dengan pendekatan monolitik (aplikasi terbungkus dalam satu *package* besar, dimana perubahan pada salah satu bagian kode program akan besar pengaruhnya terhadap kode program yang lainnya) digeser menjadi pendekatan terdistribusi. Aplikasi dibagi menjadi bagian-bagian kecil yang berfungsi spesifik (*high cohesion*) dan tidak bergantung pada komponen program lainnya (*loose coupling*), dengan antarmuka API (*Application Programming Interface*) [20].

Dari sudut pandang paradigma *microservices*, sebuah konsep kunci dalam *resilient engineering* adalah penyekatan. Jika salah satu komponen dari sistem gagal, kegagalan tersebut tidak akan *cascade* / memberikan pengaruh ke kinerja komponen yang lain. Dengan demikian, masalah dapat terisolasi dan sisanya dari komponen-komponen sistem yang lain dapat terus bekerja. Dalam sistem monolitik, jika sebuah layanan gagal, maka semuanya berhenti bekerja. Sistem dapat dijalankan pada beberapa mesin yang redundan untuk mengurangi kesempatan kegagalan/*system failure*. Sebaliknya, dengan *microservices* dapat dibangun sistem yang bisa menangani kegagalan total layanan karena layanan fungsionalitas sistem telah tersekat dalam batas yang jelas [21].

Konsep *microservices* ini sedikit banyak berbeda dengan paradigma pendahulunya, yaitu *System Oriented Architecture* (SOA). Mulai dari penggunaan protokol komunikasi pesan yang lebih ringkas (REST, dll) dibandingkan dengan SOA yang banyak menggunakan XML SOAP, sampai dengan proses desain dan pembagian fungsional yang mengedepankan pembagian berdasarkan domain fungsionalitas pada organisasi [22].

2.3 Docker (Container)

Docker adalah metode yang relatif baru untuk virtualisasi, yang tersedia native untuk 64-bit Linux. Dibandingkan dengan teknik virtualisasi yang lebih tradisional (seperti VMWare), *Docker* lebih ringan pada sumber daya sistem, dan menawarkan sistem yang hampir serupa dengan git seperti *commit* dan *tag*, dan dapat digunakan pada laptop anda maupun pada infrastruktur *cloud* [23].

Banyak keuntungan yang ditawarkan pada pendekatan kontainer yang ditawarkan oleh *Docker*, seperti memudahkan *developer* aplikasi untuk bebas dari permasalahan konfigurasi *environment*. Ketika tahap *development*, *developer* dapat menggunakan *Docker* untuk menciptakan *environment* untuk target sistem tujuan *deployment* aplikasi. *Image* dari *docker* tersebut dapat dengan mudah di *share* kepada *developer* lain pada tim pengembang, untuk bisa bekerja pada *environment* yang serupa. Ketika proses pengembangan kode program selesai, dan *software* siap di *deploy* di *server* / *cloud*, *image docker* tersebut bisa dengan mudah di *deploy* di server, sehingga permasalahan setting konfigurasi server menjadi bukan permasalahan lagi [24].

Beberapa penelitian yang mengevaluasi performa *Docker* dibandingkan sistem sejenis (*OpenStack*, virtualisasi KVM dll) menyimpulkan bahwa *Docker* memberikan performa yang superior, dengan ukuran *size image* yang kecil dan ringan tidak membebani performa *server* [25] [26] [27] [28].

Salah satu kasus penggunaan terbesar dan terkuat untuk penerapan kontainer (*Docker*) adalah *microservices*. *Microservices* adalah cara mengembangkan dan menyusun sistem perangkat lunak seperti mereka dibangun dari komponen independen kecil yang berinteraksi satu sama lain melalui jaringan. Hal ini berbeda dengan cara monolitik tradisional mengembangkan perangkat lunak, di mana ada program tunggal yang besar, misal program yang ditulis dalam bahasa pemrograman C ++ atau Java. Ketika datang kebutuhan untuk meningkatkan performa pada sistem monolit, umumnya satu-satunya pilihan adalah untuk meningkatkan keatas (*scale up*), di mana permintaan tambahan ditangani dengan menggunakan mesin yang lebih besar dengan lebih banyak RAM dan *power* CPU. Sebaliknya, *microservices* dirancang untuk skala keluar (*scale out*), di mana permintaan tambahan performa dapat ditangani oleh penyediaan beberapa tambahan mesin virtual untuk menyebar beban kerja sistem.

Dalam hal kompleksitas, *microservices* bagaikan pedang bermata dua. Pada sistem yang terdiri dari puluhan atau ratusan layanan *microservices*, kompleksitas keseluruhan sistem menjadi meningkat karena semakin rumitnya interaksi antara komponen individu. Dibandingkan dengan VMS, kontainer jauh lebih kecil dan lebih cepat untuk *deployment*, yang memungkinkan arsitektur *MICROSERVICE* menggunakan sumber daya minimal dan bereaksi dengan cepat terhadap perubahan permintaan [29].

2.4 Studi Hasil Penelitian Sebelumnya (*State of the Art*)

Kontribusi penelitian pada penelitian ini adalah implementasi *microservice* untuk studi kasus sistem manajemen asosiasi/keanggotaan. Untuk bisa menjelaskan kontribusi di atas dan posisi dari rencana penelitian ini, berikut ini di paparkan *state of the art* penelitian 5 tahun terakhir mengenai *Microservices*.

2.4.1 Design dan Pengembangan Sistem *Microservices*

[30] memilih *microservices* sebagai paradigma arsitektur dari sistem *tracking* dan manajemen armada/logistik, agar dapat meniru layanan granular untuk meningkatkan skalabilitas, dan dengan mudah mengganti layanan usang. Dilaporkan terjadinya peningkatan efisiensi operasional dan kualitas kode program karena dimungkinkan untuk menyebarkan dengan cepat layanan baru. *Microservice* juga telah diterapkan untuk mengembangkan sistem *Internet of Things / Mobile*, seperti yang dilaporkan pada [31] dan juga [32]. Dengan menggunakan studi kasus pada aplikasi *e-commerce*, [15] melaporkan meningkatkan positif pada aspek skalabilitas dari sistem *e-commerce* yang berbasis *microservices* tersebut. Implementasi *microservice* pada *smart-grid* untuk metering penggunaan listrik, dilaporkan pada paper [33]. Dari paparan diatas, dapat dilihat belum adanya laporan penelitian untuk implementasi *microservice* pada studi kasus *software* manajemen asosiasi/keanggotaan.

2.4.2 Migrasi/*Refactoring* Monolitik Sistem Menjadi Sistem yang Berbasis *Microservices*

Referensi penelitian terkini yang paling relevan dengan rencana penelitian ini adalah pada [34]. Dimana berdasarkan pengalaman melakukan migrasi sistem ke *microservice*, para peneliti di paper tersebut menyarankan sebuah metode untuk migrasi dari sistem berbasis monolitik menjadi sistem berbasis *microservices* melalui 15 langkah (*Microservices Migration Patterns*). Penelitian ini mencoba mengadopsi metode migrasi yang diusulkan [34], dan mengevaluasinya setelah selesai pengerjaan *project refactoring software* asosiasi/manajemen keanggotaan menjadi sistem yang berbasis *microservices*.

Langkah-langkah dari *Microservices Migration Patterns* adalah

1. *Enable the Continuous Integration*
2. *Recover the Current Architecture*
3. *Decompose the Monolith*
4. *Decompose the Monolith Based on Data Ownership*
5. *Change Code Dependency to Service Call*
6. *Introduce Service Discovery*
7. *Introduce Service Discovery Client*
8. *Introduce Internal Load Balancer*
9. *Introduce External Load Balancer*
10. *Introduce Circuit Breaker*
11. *Introduce Configuration Server*
12. *Introduce Edge Server*
13. *Containerize the Services*
14. *Deploy into a Cluster and Orchestrate Containers*
15. *Monitor the System and Provide Feedback*

2.4.3 *Semantic*

[35] mengembangkan sistem bernama Synapse, sistem berbasis semantik untuk *data-driven* fitur pada web, seperti *targeting customer*, *recommendations*, atau *predictions*. Puluhan *microservices* di-deploy untuk melayani 450.000 *users*. Teknologi semantik juga diterapkan pada penelitian [36] untuk memungkinkan adanya kolaborasi penggunaan *microservices* dengan organisasi luar. Dengan teknologi

semantik, dibuatlah ekosistem dimana seluruh *microservices* berkolaborasi menjadi sebuah unit yang tunggal.

2.4.4 Testing

[1] melakukan *benchmark* untuk mengukur performa sistem *microservice* yang dibangun dengan dua jenis arsitektur menggunakan *container*, yaitu: *master-slave*, atau *nested-container*. Pada paper [37], juga dilaporkan hasil pengukuran performa sistem *microservice*, namun dari sisi pengaruhnya terhadap *network traffic*. [38] mengusulkan *reusable automated acceptance testing architecture*, untuk pengujian/mengukur penerimaan pengguna (*user acceptance*) pada *project* yang dikembangkan menggunakan *microservice* dengan *Behavior-Driven Development (BDD) agile software development processes*. Di sisi lain, validasi *microservice* menjadi sorotan pada penelitian [39] dan [40]. Pada paper tersebut, diusulkan metodologi validasi untuk sistem *microservice*, dimana diperlukan validasi yang komprehensif untuk individual *microservices* dan komunikasinya dengan *microservices* lain. Pentingnya *testing* untuk menguji kualitas sistem juga turut menjadi fokus di [41], dimana dibahas penerapan *Learning-based testing (LBT)* untuk secara otomatis men-generate *test cases* untuk *blackbox testing*.

2.4.5 Resilient / Failure

Docker juga digunakan pada [40] untuk memecahkan masalah *service discovery* pada *microservice*. *Serf node* adalah *Docker image* yang bisa di-deploy ke dalam *cluster*, dimana dia kemudian menyediakan mekanisme *service discovery*, *monitoring*, dan *self-healing*. [42] juga turut membahas mengenai *self-healing* pada *microservice*, dimana sistem mampu mengelola dirinya sendiri supaya mampu beradaptasi terkait adanya lonjakan *traffic/request* maupun kegagalan/*failure* pada *microservice*.

2.4.6 Service Deployment/Discovery

Karena filosofi tersebar/terdistribusi pada *microservice* menimbulkan tantangan pada koordinasi/*discovery* layanan, peneliti dari University of Prince Edward Island [43] melaporkan penggunaan Traverna (*open source domain-independent Workflow Management System*) pada *project* yang mereka kerjakan. Pada sisi lain, [44] berusaha untuk memecahkan permasalahan yang sering timbul pada sistem *microservice*, yaitu pada konfigurasi *deployment*. Pada paper tersebut, diusulkan sebuah *tool* bernama JRO (*Jolie Redeployment Optimiser*), untuk otomatisasi dan optimasi *deployment* pada *microservices*. *Workflow microservice* yang menggunakan tool JRO tersebut dilaporkan di [45].

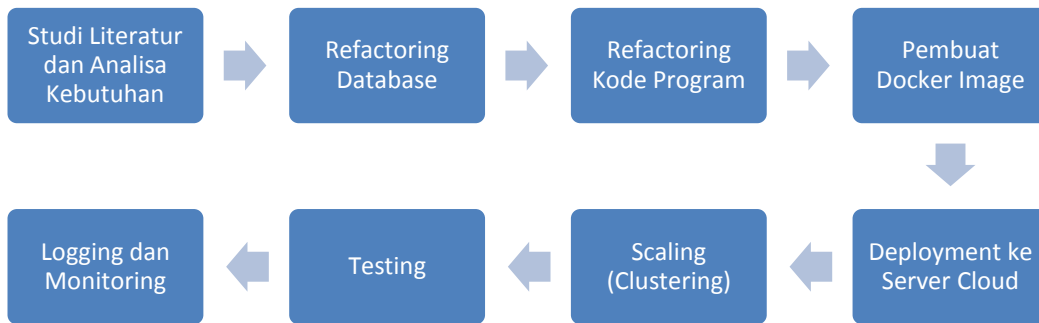
2.4.7 Business Process Modelling (BPM)

[46] mengusulkan *compiler architecture* untuk memodelkan dan mengeksekusi proses bisnis, sebagai sebuah rangkaian *microservices* yang saling berhubungan.

2.4.8 Security

Microservices sebagai sebuah arsitektur sistem yang terdistribusi menimbulkan resiko munculnya celah keamanan, pada komunikasi antar *microservices*. [47] mengusulkan sebuah *security-as-a-service* untuk aplikasi berbasis *microservices*. dengan menambahkan sebuah API pada *network hypervisor*, memungkinkan untuk adanya *flexible monitoring* dan *policy enforcement infrastructure* untuk mengamankan *network traffic* pada sistem antar *microservices*.

3. Metodologi



Gambar 1 Metode Penelitian

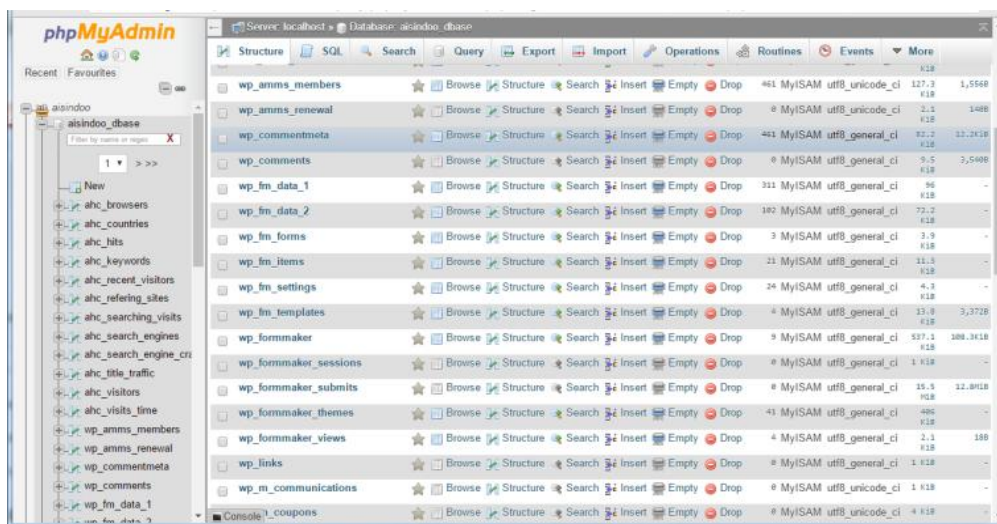
3.1 Studi Literatur dan Analisa Kebutuhan

Gambar 1 menunjukkan alur metodologi penelitian ini. Pertama-tama dilakukan studi literatur, membaca referensi dan paper terkait *Microservice / Docker Container* untuk mematangkan pemahaman terhadap permasalahan yang akan dihadapi.

3.2 Refactoring Database

Sebelum mulai menulis ulang kode program, database AISINDO yang ada saat ini (Gambar 2) di-*refactor*, untuk disesuaikan dengan rencana pengembangan menjadi sistem yang terdistribusi. Hal ini akan dilakukan dengan menjalankan tiga langkah, sesuai metode refactoring database yang dijelaskan pada [34], yaitu :

1. *Breaking Foreign Key Relationships*: memisahkan ketergantungan antar tabel, karena nantinya *foreign key look up* diganti menjadi pemanggilan API (*Application Programming Interface*) pada tabel yang dituju pada Gambar 2.
2. *Shared Static Data*: serupa dengan langkah sebelumnya, data statis yang di-*share* antar tabel akan diganti menjadi API call.
3. *Shared Tables*: pada kondisi dimana satu tabel diakses oleh beberapa domain bisnis, maka tabel perlu dipecah menjadi dua atau lebih sesuai dengan pengaturan domain kepemilikan data.



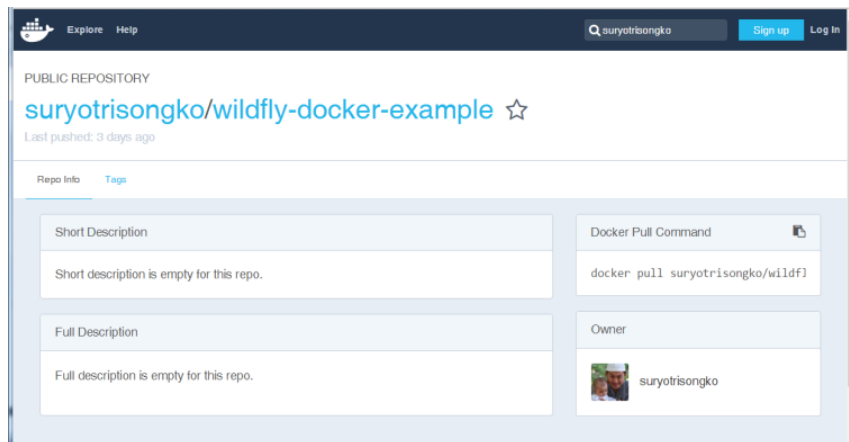
Gambar 2 Database Manajemen Asosiasi/Keanggotaan AISINDO saat ini

3.3 Refactoring Kode Program

Kode program akan di-*refactor*, dipecah berdasarkan domain fungsionalitas bisnisnya, sesuai metode refactoring database yang dijelaskan pada [34]. Kode program ditulis menggunakan bahasa pemrograman Java EE. Pola integrasi yang akan digunakan adalah menggunakan *REST Asynchronous*, namun pada kondisi dimana diperlukan adanya transaksi (ACID) maka akan menggunakan *Java Transaction API* (JTA).

3.4 Pembuat Docker Image

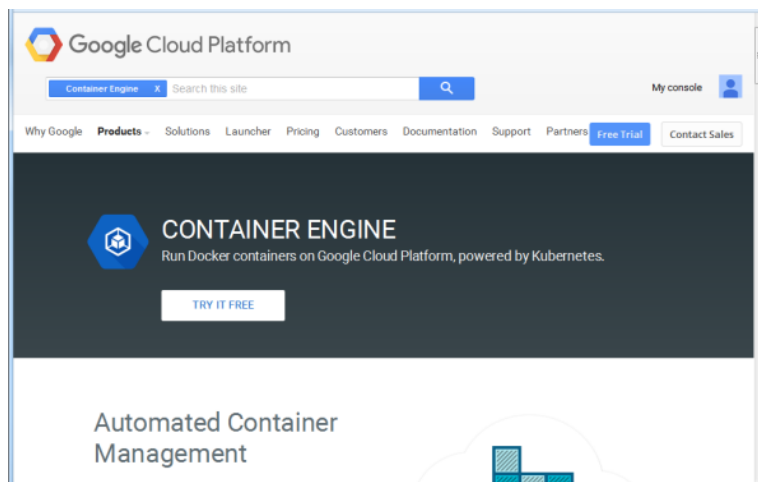
Docker image berbasis Fedora 21, *application Server Wildfly 7* dan database *Apache Cassandra* akan digunakan untuk *image container* aplikasi *microservice* yang akan digunakan (lihat Gambar 3).



Gambar 3 *Docker image*

3.5 Deployment ke Server Cloud

Deployment container Docker yang telah dikembangkan direncanakan di layanan Google App Engine atau layanan cloud yang lain, misal Amazon AWS (Gambar 4).



Gambar 4 *Google app engine*

3.6 Scaling (Clustering)

Clustering docker container akan menggunakan teknologi *Docker Swarm*. Arsitektur dasar *Swarm* cukup mudah dimana masing-masing *host* menjalankan agen *Swarm* dan satu *host* menjalankan *manajer Swarm* (*cluster* tes kecil *host* ini juga dapat menjalankan agen). Manajer bertanggung jawab untuk orkestrasi dan penjadwalan kontainer pada *host*.

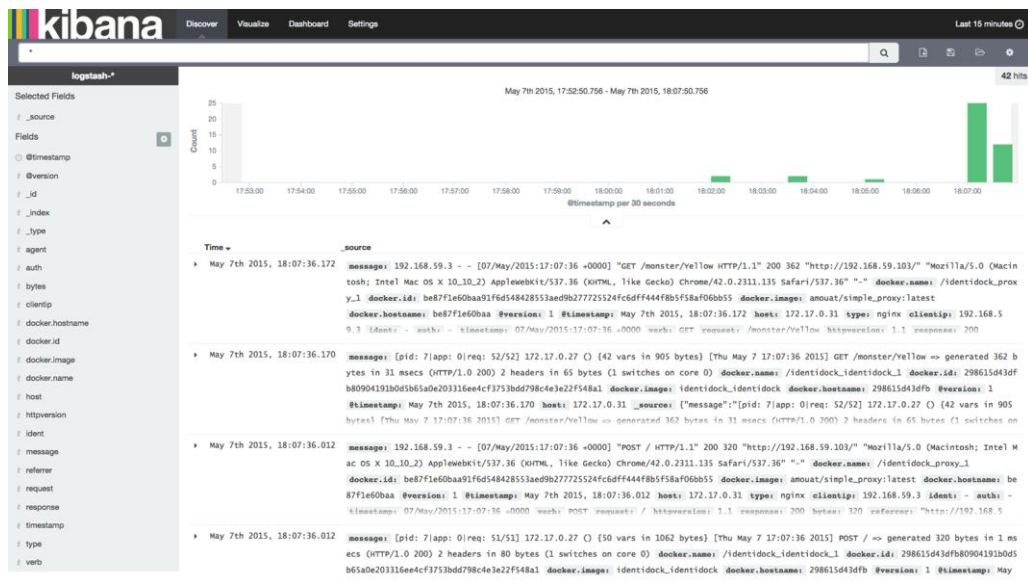
3.7 Testing

Tiga macam pengujian akan dilakukan pada sistem *microservices* yang sudah dikembangkan yaitu:

1. *Unit test*: Setiap *service* harus memiliki seperangkat *unit test* yang terkait. *Unit test* harus menguji potongan kecil dan terisolasi dari fungsi.
2. *Component tests* : Ini dapat pada tingkat pengujian antarmuka eksternal dari layanan individual, atau pada tingkat pengujian subsistem dari kelompok *service*. Pengujian ini juga penting untuk mengekspos metrik dan *logging* melalui layanan API pada saat pengujian.
3. *End-to-end test* : Tes yang memastikan seluruh sistem bekerja. Semua fungsionalitas bisnis yang dirancang diuji kinerjanya.

3.8 Logging dan Monitoring

Untuk *logging* dan *monitoring* sistem, ELK stack (Elasticsearch, Logstash, and Kibana) akan digunakan (lihat Gambar 5).



Gambar 5 Kibana

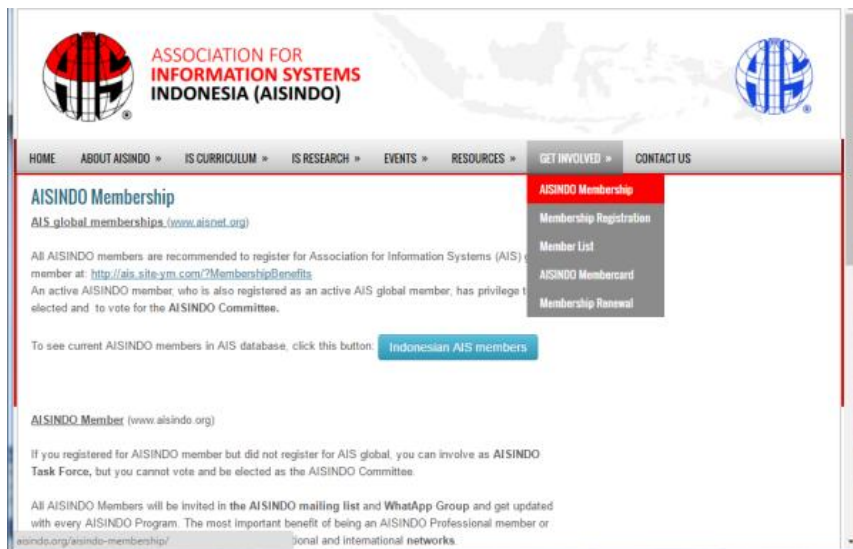
4. Hasil dan Pembahasan

4.1 Studi literatur dan analisa kebutuhan

Pertama-tama, dilakukan analisa kebutuhan yang lebih mendalam, mulai dari dengan cara mengevaluasi sistem sebelumnya (Gambar 6 dan 7), hingga wawancara dengan berbagai *stakeholder* AISINDO terlebih dahulu.



Gambar 6 *Software* hasil penelitian sebelumnya: *Open Source* manajemen asosiasi/keanggotaan



Gambar 7 *Software* hasil penelitian sebelumnya telah digunakan untuk manajemen anggota AISINDO (Asosiasi Profesi Sistem Informasi Indonesia)

Selain mempelajari berbagai paper "*state of the art*" tentang topik yang terkait, juga dilakukan telaah pada buku practical untuk persiapan teknis penulisan kode program dan database. Kemudian, untuk bisa memahami kebutuhan sistem pada AISINDO, maka dilakukan kajian analisa kebutuhan sistem, dengan cara menganalisa sistem eksisting dan juga wawancara pada *stakeholder* dan pemegang keputusan di organisasi AISINDO. Wawancara dan diskusi dengan *stakeholder* dilakukan pada even AISINDO *Annual Meeting* yang digelar di hotel Inna Garuda Yogyakarta, 14 Agustus 2016.

Selanjutnya, dilakukan analisa kondisi sistem eksisting, untuk memetakan *Use Case* kebutuhan sistem yang ada. *Use Case* dibagi menjadi dua bagian, yaitu *Use Case* untuk pengguna tamu (pengunjung web aisindo/anggota asosiasi AISINDO) dan *Use Case* untuk admin sistem (direktur keanggotaan AISINDO

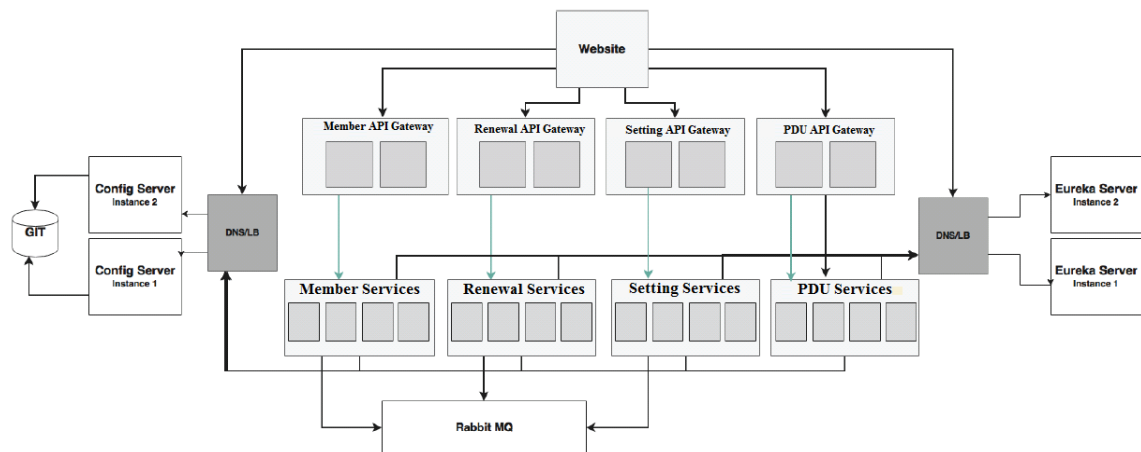
dan tim). Pada masing-masing kondisi eksisting, kemudian disimpulkan kebutuhan *service/layanan* yang harus ada pada sistem baru.

4.2 Refactoring Database

Pada tahap ini, dilakukan analisa kondisi *existing* dan kemudian dilakukan transformasi yang diperlukan untuk menyiapkan *database* pada sistem baru. Pada sistem *existing*, *database* menggunakan database MySQL. Hasil dari *refactoring database* berupa struktur *database* NoSQL Cassandra, yang memiliki perbedaan cara kerja dibandingkan dengan Relational DBMS seperti dihilangkannya *foreign key constraint*, dan sebagainya.

4.3 Refactoring Kode Program

Kode program dikembangkan menggunakan Java Spring Boot, dengan arsitektur dapat dilihat pada Gambar 8 yang menggambarkan implementasi Arsitektur *Microservice* untuk Resiliensi Sistem Informasi. Gambar 8 juga menunjukkan letak dari Arsitektur *Microservice* yang berfungsi untuk Resiliensi Sistem Informasi, yaitu pada *microservice node* yang terbungkus oleh *Zuul proxy* sebagai *Gateway sistem load balancer*, dengan *Eureka server* dan *Config server* untuk memungkinkan manajemen sistem yang terpusat dan dinamis.



Gambar 8. Arsitektur Sistem

Keterangan :

- 1) *Spring Cloud Config*. "*Spring Cloud Config*" menyediakan dukungan *server* dan *client-side* untuk konfigurasi *externalized* dalam sistem terdistribusi. Dengan "*Config Server*" sistem ini memiliki tempat sentral untuk mengelola properti eksternal untuk aplikasi di semua lingkungan. Implementasi standar dari *backend server* penyimpanan menggunakan "*git*".
- 2) *EUREKA*. *EUREKA* untuk *Service registry*: Sebuah registri layanan menyediakan lingkungan *runtime* untuk layanan untuk secara otomatis mempublikasikan ketersediaan mereka pada saat *runtime*. Sebuah registri akan menjadi sumber informasi yang baik untuk memahami topologi layanan di setiap titik.
- 3) *ZUUL Proxy*. *Zuul Proxy* secara internal menggunakan *server Eureka* untuk *service discovery*, dan *Ribbon* untuk *load balancing* antara layanan. *Proxy Zuul* juga mampu melakukan *routing*, *monitoring*, mengelola ketahanan, keamanan, dan sebagainya. Secara sederhana, dapat dipertimbangkan *Zuul* layanan *reverse proxy*. Dengan *Zuul*, bahkan dapat diubah perilaku dari layanan mendasari dengan menimpa mereka di lapisan API.

4.3.1 Hasil Kode Program *BackEnd*

Microservice yang berjalan pada sisi belakang (*back-end*) meliputi *REST-Service* yang dibagi menjadi dua bagian: 1) *REST Service* yang terbuka dan tidak memerlukan otentifikasi, dan 2) *REST Service* yang terproteksi oleh *JWT Token Based Otentifikasi*. Pola komunikasi *front-end* dengan *back-end* terjadi secara *asynchronous*, sehingga *server* tidak perlu lagi mengingat *session* sebagaimana seperti pola komunikasi *client-server* untuk aplikasi berbasis web seperti biasanya. Berikut ini daftar fungsi REST yang tersedia pada sisi *back-end*, dapat dilihat di Gambar 9 dan Gambar 10. Format data JSON digunakan untuk memudahkan/menghilangkan beban *payload* untuk melakukan *parsing* pesan.

expert-controller : Expert Controller			Show/Hide	List Operations	Expand Operations
GET	/expert/all	getAllExperts			
GET	/expert/detail	getExpertById			
GET	/expert/featured	getFeaturedExperts			
GET	/expert/search	getExpertByKeywordExpertiseLocation			
pdu-resources : Pdu Resources			Show/Hide	List Operations	Expand Operations
photo-resource : Photo Resource			Show/Hide	List Operations	Expand Operations
specialization-controller : Specialization Controller			Show/Hide	List Operations	Expand Operations
GET	/specialization/all	getAllSpecializations			
GET	/specialization/detail	getSpecializationById			
user-controller : User Controller			Show/Hide	List Operations	Expand Operations
GET	/user/isactivemember	isactivemember			
POST	/user/login	login			
POST	/user/register	registerUser			
POST	/user/resetpassword	resetpassword			
GET	/user/verify	verifyUser			

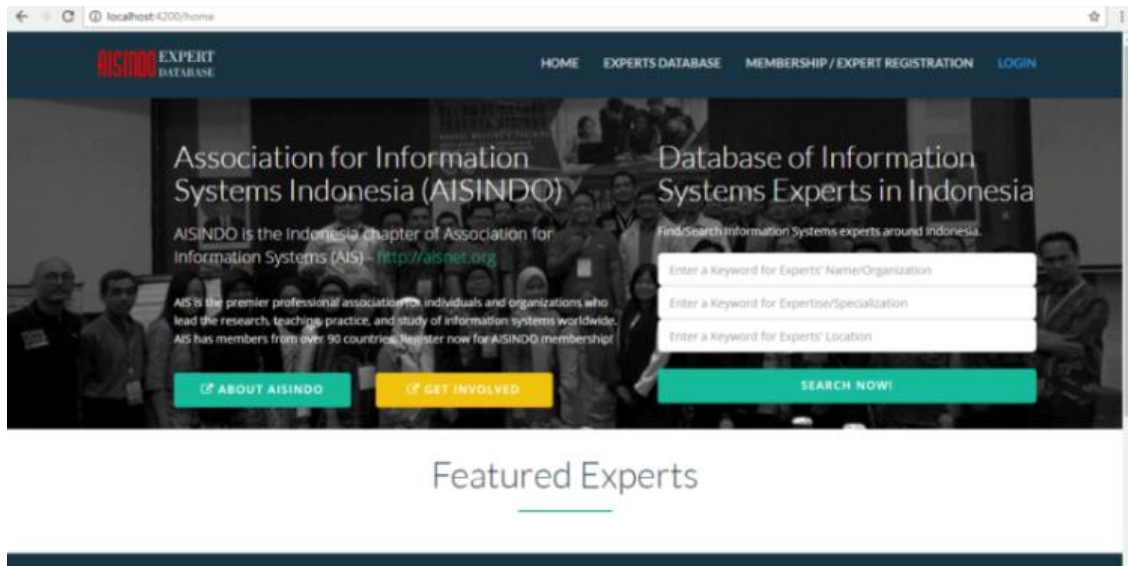
Gambar 9 REST Service

basic-error-controller : Basic Error Controller			Show/Hide	List Operations	Expand Operations
expert-controller : Expert Controller			Show/Hide	List Operations	Expand Operations
pdu-resources : Pdu Resources			Show/Hide	List Operations	Expand Operations
GET	/rest/pdu/all	allPdu			
POST	/rest/pdu/detailbyemail	detailPduByEmail			
POST	/rest/pdu/update	updatePdu			
photo-resource : Photo Resource			Show/Hide	List Operations	Expand Operations
POST	/rest/photo/upload	upload			
specialization-controller : Specialization Controller			Show/Hide	List Operations	Expand Operations
user-controller : User Controller			Show/Hide	List Operations	Expand Operations
user-resources : User Resources			Show/Hide	List Operations	Expand Operations
GET	/rest/user/all	allUser			
POST	/rest/user/detailbyemail	detailUserByEmail			
GET	/rest/user/search	searchUser			
POST	/rest/user/update	updateUser			
GET	/rest/user/users	loginSuccess			

Gambar 10 REST Security dengan JWT Token Based Authentication

4.3.2 Hasil kode program *Front End*

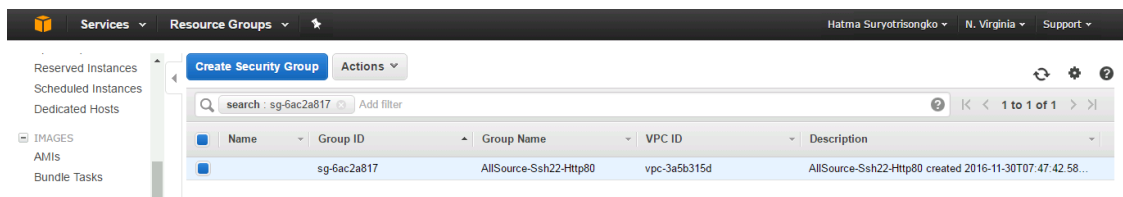
Kode program *Front End* di-develop menggunakan *angular 2 typescript*. Terdiri dari beberapa fasilitas utama, seperti: fitur pencarian *expert*, fitur *featured expert*, fitur registrasi *membership*, dan fitur klain PDU (*Personal Development Unit*) untuk *member* bisa meningkatkan level status *membership*-nya. *Screenshot* aplikasi dapat dilihat pada Gambar 11.



Gambar 11 *Front end* halaman depan AISINDO *Membership / Expert Database*

4.4 Pembuat *Docker Image*, *Deployment ke Server Cloud*, dan *Scaling (Clustering)*

Setelah *front-end* dan *back-end* selesai dikerjakan *microservices* di-deploy ke *server hosting* akan menggunakan AWS Amazon EC2 (Gambar 12). Prototipe sistem disiapkan menggunakan lima komputer *server* yang sudah disiapkan di laboratorium, untuk menguji sistem secara lokal.



Gambar 12 Amazon AWS EC2 *Instance*

4.5 *Testing, Logging dan Monitoring*

Selain pengujian standar seperti *Unit Testing*, *Component Testing* dan *End to End Testing*, juga dilakukan pengujian untuk spesifik menguji apakah resiliensi sudah tercapai. Pengujian dilakukan untuk menguji apakah arsitektur sistem yang diusulkan pada Gambar 8 tetap dapat *available* ketika beberapa *service node* mengalami gangguan. Simulasi kegagalan/gangguan dilakukan dengan cara mensimulasikan *system down/failure* secara acak pada *node-node microservice*. Hasil dari pengujian tersebut menyimpulkan bahwa layanan *microservice* tetap dapat selalu tersedia, meskipun salah satu *node* dimatikan secara acak, selama masih ada minimum satu *node microservice* yang tersedia.

5. Kesimpulan

5.1 Simpulan

Model/desain arsitektur *software*/sistem informasi yang telah dikembangkan menunjukkan aspek kualitas resiliensi, dengan pendekatan *Microservices - Docker container* yang telah dikembangkan dengan sistem *back-end Spring Boot* dan *front-end Angular2*. Pola komunikasi *REST Service* dengan format JSON memberikan kemudahan dalam implementasi dan juga tidak membebani *server*.

Proof of concept dari model/desain arsitektur yang diusulkan telah dibuat, dengan menulis ulang kode program (*refactoring*) *Software Open Source* manajemen asosiasi/keanggotaan. Kode program tersebut sudah diimplementasikan pada website AISINDO (Asosiasi Profesi Sistem Informasi Indonesia) di alamat web expert.aisindo.org, sehingga untuk kelanjutan penelitian berikutnya bisa memanfaatkan data empiris untuk evaluasi dan perbaikan model di penelitian berikutnya.

Aspek resiliensi dapat diamati pada ketangguhan system ketika salah satu *instance microservice* mengalami gangguan. Dari sudut pandang *end-user*, gangguan ini tidak akan dapat dirasakan, karena *back-end* telah ditangani oleh kombinasi antara *Eureka* dan *Zuul proxy* yang akan berfungsi sebagai *load balancer* untuk membagi beban sekaligus untuk menyembunyikan gangguan yang dialami oleh sistem di belakang *layer*. Dari hasil pengujian, dapat disimpulkan bahwa aspek kualitas resiliensi dapat dicapai pada arsitektur sistem yang diusulkan di penelitian ini.

Penelitian ini berhasil menunjukkan implementasi *microservice* untuk studi kasus sistem manajemen asosiasi/keanggotaan yang digunakan untuk membuktikan adopsi model *Microservices Migration Patterns*, lalu mengevaluasi model tersebut pada implementasi di Asosiasi Sistem Informasi Indonesia (AISINDO).

5.2 Saran

Untuk penelitian berikutnya, model *microservice* yang digunakan pada penelitian ini sebaiknya dimodifikasi dengan memanfaatkan model *Hybrid Cloud*. Pola model ini akan sangat cocok untuk kasus penggunaan dimana *database* sangat membutuhkan *privacy*, sebagai contoh data *public* pada *e-Government* di Indonesia.

6. Pengakuan

Penelitian ini dibiayai oleh skema Hibah Penelitian Pemula ITS 2016.

7. Daftar Rujukan

- [1] Müller, G., Koslowski, T. G., & Accorsi, R. (2013). *Resilience - A New Research Field in Business Information Systems?* In W. Abramowicz (Ed.), *Business Information Systems Workshops* (pp. 3–14). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-41687-3_2
- [2] Zhang, P. W. J. (2010). Guest Editor's foreword. *Enterprise Information Systems*, 4(2), 95–97. <http://doi.org/10.1080/17517571003770468>
- [3] Liu, D., Deters, R., & Zhang, W. J. (2010). *Architectural design for resilience*. *Enterprise Information Systems*, 4(2), 137–152. <http://doi.org/10.1080/17517570903067751>
- [4] Microservices. (n.d.). Retrieved March 30, 2016, from <http://martinfowler.com/articles/microservices.html>
- [5] Stubbs, J., Moreira, W., & Dooley, R. (2015). *Distributed Systems of Microservices Using Docker and Serfnode*. In 2015 7th International Workshop on Science Gateways (IWSG) (pp. 34–39). <http://doi.org/10.1109/IWSG.2015.16>
- [6] Bhamra, R., Dani, S., & Burnard, K. (2011). *Resilience: the concept, a literature review and future directions*. *International Journal of Production Research*, 49(18), 5375–5393. <http://doi.org/10.1080/00207543.2011.563826>
- [7] Ishida, Y. (2015). *Self-Repair Networks: A Mechanism Design* (Vol. 101). Springer.

- [8] Bodin, P., & Wiman, B. (2004). *Resilience and other stability concepts in ecology: Notes on their origin, validity, and usefulness*. ESS Bulletin, 2(2), 33–43.
- [9] Holling, C. S. (1973). *Resilience and stability of ecological systems*. Annual Review of Ecology and Systematics, 1–23.
- [10] Walker1a, B., Carpenter, S., Anderies1b, J., Abell1b, N., Cumming, G., Janssen, M., ... Pritchard, R. (2002). *Resilience management in social-ecological systems: a working hypothesis for a participatory approach*. Conservation Ecology, 6(1), 14.
- [11] Gunderson, L. H. (2000). *Ecological resilience--in theory and application*. Annual Review of Ecology and Systematics, 425–439.
- [12] Tilman, D., & Downing, J. A. (1996). *Biodiversity and stability in grasslands*. In Ecosystem Management (pp. 3–7). Springer.
- [13] Carpenter, S., Walker, B., Anderies, J. M., & Abel, N. (2001). *From metaphor to measurement: resilience of what to what?* Ecosystems, 4(8), 765–781.
- [14] Luthans, F., Vogelgesang, G. R., & Lester, P. B. (2006). *Developing the psychological capital of resiliency*. Human Resource Development Review, 5(1), 25–44.
- [15] Bruneau, M., Chang, S. E., Eguchi, R. T., Lee, G. C., O'Rourke, T. D., Reinhorn, A. M., ... von Winterfeldt, D. (2003). *A framework to quantitatively assess and enhance the seismic resilience of communities*. Earthquake Spectra, 19(4), 733–752.
- [16] Hasselbring, W. (2016). *Microservices for Scalability* (Keynote Presentation).
- [17] Coutu, D. L. (2002). *How resilience works*. Harvard Business Review, 80(5), 46–56.
- [18] Hamel, G., & Valikangas, L. (2003). *The quest for resilience*. Harvard Business Review, 81(9), 52–65.
- [19] Hollnagel, E., Woods, D. D., & Leveson, N. (2007). *Resilience engineering: concepts and precepts*. Ashgate Publishing, Ltd..
- [20] Newman, S. (2015). *Building Microservices*. O'Reilly Media, Inc.
- [21] Namiot, D., & Sneps-Snepe, M. (2014). *On micro-services architecture*. International Journal of Open Information Technologies, 2(9).
- [22] Thones, J. (2015). *Microservices*. Software, IEEE, 32(1), 116–116.
- [23] Fink, J. (2014). *Docker: A software as a service, operating system-level virtualization framework*. Code4Lib Journal, 25.
- [24] Jiang, K., & Song, Q. (2015). *A Preliminary Investigation of Container-Based Virtualization in Information Technology Education* (pp. 149–152). Presented at the Proceedings of the 16th Annual Conference on Information Technology Education, ACM.
- [25] Ismail, B. I., Mostajeran Goortani, E., Ab Karim, M. B., Ming Tat, W., Setapa, S., Luke, J. Y., & Hong Hoe, O. (2015). *Evaluation of Docker as Edge computing platform* (pp. 130–135). Presented at the Open Systems (ICOS), 2015 IEEE Conference on, IEEE.
- [26] Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M., & Steinder, M. (2015). *Performance Evaluation of Microservices Architectures using Containers* (pp. 27–34). Presented at the Network Computing and Applications (NCA), 2015 IEEE 14th International Symposium on, IEEE.
- [27] Špacek, F., Sohlich, R., & Dulík, T. (2015). *Docker as platform for assignments evaluation*. Procedia Engineering, 100, 1665–1671.
- [28] Ou, A. Y.-Z., & Chen, J.-C. (2015). *Head-to-Head: Which is the Better Cloud Platform for Early Stage Start-up? Docker versus OpenStack*.
- [29] Mouat, A. (2015). *Using Docker: Developing and Deploying Software with Containers*. " O'Reilly Media, Inc."
- [30] Schneider, T., Car, C., & Wolfsmantel, A. (2016). *Achieving Cloud Scalability with Microservices and DevOps in the Connected Car Domain*. In Software Engineering (Workshops) (pp. 138–141).
- [31] Bak, P., Melamed, R., Moshkovich, D., Nardi, Y., Ship, H., & Yaeli, A. (2015). *Location and Context-based Microservices for Mobile and Internet of Things Workloads* (pp. 1–8). Presented at the Mobile Services (MS), 2015 IEEE International Conference on, IEEE.
- [32] Familiar, B. (2015). *IoT and Microservices*. In Microservices, IoT, and Azure (pp. 133–163). Springer.
- [33] Rodríguez Molina, J. (2015). *Distribution of microservices for hardware interoperability in the Smart Grid* (Doctoral dissertation, ETSIS_Telecomunicacion).
- [34] Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2015). *Microservices Migration Patterns*.
- [35] Viennot, N., Lécuyer, M., Bell, J., Geambasu, R., & Nieh, J. (2015). *Synapse: a microservices architecture for heterogeneous-database web applications* (p. 21). Presented at the Proceedings of the Tenth European Conference on Computer Systems, ACM.
- [36] Versteden, A., Pauwels, E., & Papantoniou, A. (2015). *An Ecosystem of User-facing Microservices Supported by Semantic Models*. (pp. 12–21). Presented at the USEWOD-PROFILES@ ESWC.
- [37] Kratzke, N. (2015). *About Microservices, Containers and their Underestimated Impact on Network Performance*. CLOUD COMPUTING 2015, 180.
- [38] Rahman, M., & Gao, J. (2015). *A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development* (pp. 321–325). Presented at the Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on, IEEE.
- [39] Savchenko, D., & Radchenko, G. (2015). *Microservices Validation: Methodology and Implementation*. Presented at the CEUR Workshop Proceedings. Vol. 1513: Proceedings of the 1st Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists (Ural-PDC 2015).—Yekaterinburg, 2015.
- [40] Savchenko, D., Radchenko, G., & Taipale, O. (2015). *Microservices validation: Mjolinrr platform case study* (pp. 235–240). Presented at the Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on, IEEE.
- [41] Nycander, P. (2015). *Learning-Based Testing of Microservices: An Exploratory Case Study Using LBTest*.
- [42] Toffetti, G., Brunner, S., Blöchliger, M., Dudouet, F., & Edmonds, A. (2015). *An architecture for self-managing microservices* (pp. 19–24). Presented at the Proceedings of the 1st International Workshop on Automated Incident Management in Cloud, ACM.

- [43] Moses, D., & Pound, P. (2014). *A Distributed Microservices Framework Integrating Taverna*. Open Repositories 2014.
- [44] Gabbrielli, M., Giallorenzo, S., Guidi, C., Mauro, J., & Montesi, F. (2016). *Self-Reconfiguring Microservices*. In *Theory and Practice of Formal Methods* (pp. 194–210). Springer.
- [45] Safina, L., Mazzara, M., & Montesi, F. (2015). *Data-driven Workflows for Microservices*. arXiv Preprint arXiv:1511.02597.
- [46] Singer, R. (2016). *Business Process Modeling and Execution--A Compiler for Distributed Microservices*. arXiv Preprint arXiv:1601.05976.
- [47] Sun, Y., Nanda, S., & Jaeger, T. (2015). *Security-as-a-Service for Microservices-Based Cloud Applications* (pp. 50–57). Presented at the 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), IEEE.