Faktor Exacta

Vol. 13, No. 3, Sept 2020, pp. 140~148
eISSN: 2502-339X, pISSN: 1979-276X, DOI: 10.30998/faktorexacta.v13i3.6426 

# Orkestrasi *Cloud* Dengan Chef, Menuju Keselarasan Sistem Otomasi Teknologi Informasi

**Suryo Bramasto[1], Melani Indriasari[2], Endang Ratnawati D.[3]**
[1,2,3]Program Studi Informatika, Institut Teknologi Indonesia, Indonesia

**Article Info**

***Article history:***

Received June 03, 2020
Revised Okt 29, 2020
Accepted Nov 06, 2020

***Keywords:***

Chef
Cloud
deployment
DevOps
portabilitas
TOSCA
orkestrasi
otomasi

**ABSTRAK**

Penelitian pada artikel ini mengimplementasikan orkestrasi dari otomasi layanan-layanan teknologi informasi berbasis *cloud*, yang mana dari sisi pengguna merupakan solusi-solusi bisnis. Orkestrasi dilakukan dengan mesin orkestrasi Chef. Layanan-layanan teknologi informasi yang *deployment* dan pengelolaannya pada *cloud* terotomasi dengan orkestrasi Chef, dibangun dengan *web server* Apache, *database server* MySQL, dan *load balancer* Nginx. Orkestrasi dilakukan dengan membangun arsitektur orkestrasi Chef spesifik layanan cloud AWS (*Amazon Web Services*), kemudian membuat *recipes* dan *cookbooks* pada Chef yang berisi perintah-perintah otomasi dan orkestrasi layanan-layanan teknologi informasi. Orkestrasi memungkinkan sebagian besar pekerjaan dari system administrator untuk dilakukan secara otomatis bahkan pada layanan-layanan IT berbasis Cloud yang kompleks, dengan tetap memungkinkan untuk dilakukan pengelolaan layanan sebagaimana yang biasa dilakukan oleh *system administrator* pada sub-sub sistem dari layanan-layanan IT. Selain itu orkestrasi pada implementasinya akan sangat mendukung pendekatan DevOps pada rancang bangun piranti lunak sebagai sub sistem utama dari layanan IT. Manfaat orkestrasi Chef pada artikel ini adalah pada dua contoh kasus. Yang pertama yakni memungkinkan tim DevOps melakukan eksekusi tugas secara paralel atau tidak saling menunggu dalam *deployment* empat web services berbasis LAMP (Linux-Apache-MySQL-Python) *web service stack* secara bersamaan sebagai Software as a Service (SaaS) pada AWS. Sedangkan yang kedua yakni mewujudkan portabilitas pada aplikasi *cloud multi-tier* berbasis OpenStack dan Opscode Chef, sesuai standar Topology and Orchestration Specification for Cloud Application (TOSCA) versi 1.



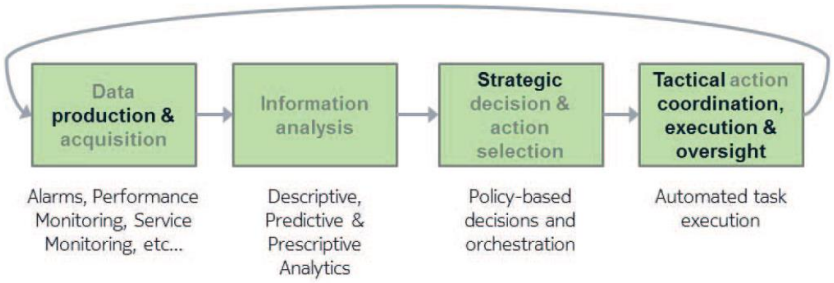
***Corresponding Author:***

Suryo Bramasto,
Program Studi Informatika,
Institut Teknologi Indonesia,
Jl. Raya Puspiptek Serpong, Tangerang Selatan.
Email: suryo.bramasto@iti.ac.id

## 1. PENDAHULUAN

Otomasi memungkinkan replikasi dan perulangan tugas-tugas Teknologi Informasi (TI) tradisional menggunakan layanan *cloud*. Orkestrasi merupakan bagian dari *service-oriented architectures*, dimana merupakan teknologi yang memungkinkan sejumlah *loosely coupled services* dikolaborasikan secara harmonis menjadi workflow yang lebih besar [1]. Interaksi dari *loosely coupled services* tersebut juga harus terkoordinasi dalam konteks workflow secara terbuka dan *platform-independent*. Orkestrasi memungkinkan pengaturan, pengurutan, dan koordinasi tugas-tugas terotomasi. Tiga macam otomasi yang biasa dilakukan pada ranah *cloud* yakni menyediakan server-server virtual sesuai dengan spesifikasi *client*, *data backup*, dan *clean-up* pada ranah sistem-sistem virtual dan atau *cloud instances*. Model dasar otomasi

*cloud* ditunjukkan pada gambar 1. Orkestrasi meletakkan tugas-tugas terotomasi pada *workflow* yang sesuai dan memenuhi kebutuhan bisnis, yang mana dapat mencakup *scaled servers* terotomasi, *data backup*, dan *clean-up*. Orkestrasi tidak fokus pada tugas-tugas karena merupakan ranah otomasi. Orkestrasi fokus pada *workflow* dan koordinasi. Hubungan antara orksetrasi dengan otomasi terdefinisi dalam dua tingkat seperti ditunjukkan pada tabel 1. Kedua tingkat tersebut harus bekerja sama dan berkolaborasi secara lancar dan harmonis guna kelancaran teknis dan proses bisnis organisasi. Dengan demikian dibutuhkan suatu mesin orkestrasi, yang mana salah satunya adalah Chef. Otomasi dan orkestrasi merupakan prioritas utama pada pendekatan DevOps terhadap rekayasa piranti lunak. DevOps merupakan kumpulan cara yang mengkombinasikan *software development* (Dev) dan *information technology (IT) operations* (Ops) yang bertujuan mempersingkat *system development life cycle* (SDLC) dan menyediakan penghantaran berkelanjutan dengan kualitas piranti lunak yang tinggi [2]. Kemudian manfaat lain dari orkestrasi yang tereksplorasi baru-baru ini adalah dukungan yang memungkinkan portabilitas aplikasi *cloud* [3], dimana sebelumnya migrasi ke layanan *cloud* baru dimungkinkan untuk lapisan aplikasi dan data [4]–[6].




Gambar 1. Model dasar otomasi cloud [7]

**Tabel. 1** Hubungan antara orkestrasi dengan otomasi [8]

| Tingkat | Fungsi | Deskripsi |
|---|---|---|
| **I: Otomasi** | Pekerjaan TI, direpresentasikan sebagai *template* yang mengandung dan mereplikasi tugas-tugas individual | Mengambil sekumpulan tugas-tugas dan meletakkan pada template |
| **II: Orkestrasi** | Fungsi administratif pada *cloud* | Aktifitas administratif, mengatur dan mengkoordinir tugas-tugas untuk sebuah bisnis |

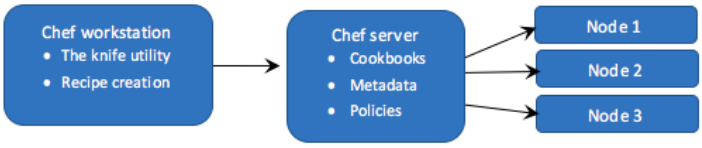
Orkestrasi terdefinisi dalam Topology and Orchestration Specification for Cloud Applications (TOSCA), yang dirumuskan oleh OASIS (Organization for the Advancement of Structured Information Standards), dan disponsori oleh IBM, CA Technologies, Hewlett-Packard, Red Hat, dan SAP [9]. Sedangkan penelitian-penelitian tentang orkestrasi dan otomasi secara spesifik dalam kaitannya dengan portabilitas aplikasi *cloud* juga telah banyak dilakukan oleh dunia industri guna muwujudkan topologi layanan *cloud* yang portabel [10], [11].

Seperti halnya mesin orkestrasi yang lain, Chef juga menyediakan *template* dan infrastruktur. Apapun piranti lunak yang terlibat, tahap pertama adalah menggunakan kode guna mendefinisikan infrastruktur yang ditentukan. Artinya menciptakan "kembaran digital" atau paling tidak *mirror* dalam bentuk kode program untuk setiap bagian dari infrastruktur yang hendak diterapkan otomasi dan orkestrasi. Misalkan telah dimiliki sumber daya utama yakni web server yang terikat dengan suatu basis data, berikut server *load balancing* yang kesemuanya ingin diotomasikan dan kemudian dilakukan orkestrasi. Dalam kasus tersebut, dibutuhkan suatu *markup language* guna mendifinisikan sumber daya.

Chef menggunakan bahasa pemrograman Ruby guna mendefinisikan *template-template*nya yang biasa disebut dengan resep. Chef menggunakan Python DSL (Domain Specific Language) untuk menciptakan resep-resep. Sangat penting untuk memahami resep guna mengetahui urutan otomasi tugas dengan Chef, dan kemudian dilakukan orkestrasi sehingga web server, basis data, dan load server berfungsi dengan benar dan kolaboratif selaras dengan tujuan organisasi. Setiap resep akan diletakkan pada domain-domain yang ada pada gambar 3. Resep pada domain-domain disebut juga dengan *cookbooks*.

Arsitektur Chef ditunjukkan pada gambar 2, dimana penjelasan dari terminologi dari arsitektur Chef tersebut adalah sebagai berikut:

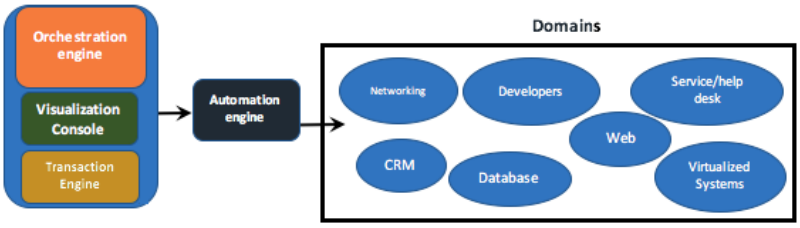
- Chef Workstation: Komputer guna konfigurasi mesin orkestrasi (Chef *server*). Chef *workstation* digunakan juga untuk mengendalikan Chef *server* dan memasukkan perintah-perintah guna membuat resep, sehingga *template* terotomasi guna menciptakan dan mendefinisikan sistem dapat dibangun. Dengan demikian selanjutnya dapat dilakukan orkestrasi terhadap *template-template* terotomasi tersebut. Chef *workstation* digunakan juga untuk konfigurasi resep (*templates*) dan meletakkan resep pada *cookbooks*.
- Chef server: Mesin orkestrasi, yang mana merupakan bagian utama dari sistem dan menyimpan semua informasi terkait otomasi dan orkestrasi.
- Chef node: *Server* atau sistem yang dikelola dengan Chef *server*




Gambar 2. Infrastruktur Chef

## 2. METODE

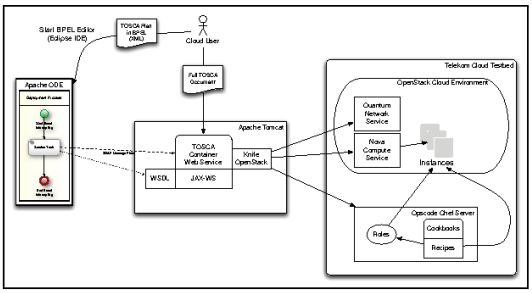
Arsitektur orkestrasi *cloud* menggunakan model yang ditunjukkan pada gambar 3.



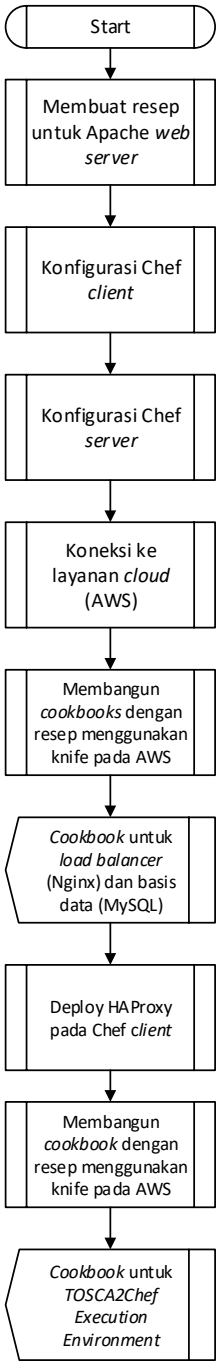

Gambar 3. Model orkestrasi *cloud* [8]

Terkait dengan proses orkestrasi menggunakan mesin yang menyediakan *template*, *domain Service/help desk* yang ditunjukkan pada gambar 3, memegang peranan penting yakni konfigurasi *end point* tertentu. Model orekstrasi *cloud* ini menggunakan sebuah agen/manajer. Agen-agen tersebar pada berbagai area di dalam organisasi, sedangkan manajer berada pada *server* orkestrasi. Dimungkinkan juga untuk ditambahkan beberapa manajer bagian yang diletakkan pada *server* tambahan pada implementasi dengan skala lebih besar, dimana semua manajer bagian berkomunikasi dan dikoordinir langsung oleh manajer.

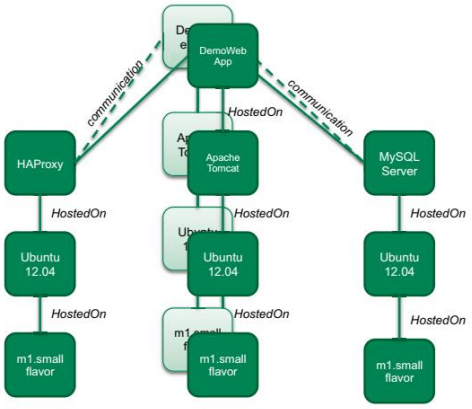
Dalam mewujudkan protabilitas pada aplikasi *cloud multi-tier* sesuai standar TOSCA, model orkestrasi *cloud* yang ditunjukkan pada gambar 3 dibangun dengan memenuhi arsitektur lingkungan eksekusi TOSCA2Chef. Secara spesifik, orkestrasi Chef bertujuan mewujudkan fitur aktual portabilitas dari TOSCA, dimana mulai dari fase pemodelan *use case* hingga terwujudnya suatu *web service* secara keseluruhan berjalan pada cloud. TOSCA mendefinisikan model proses untuk diimplementasikan dengan knife dalam Business Process Execution Language (BPEL) [12]. Pada penelitian ini, BPEL dibangun di dalam model orkestrasi *cloud* Apache Orchestration Director Engine (ODE) [13]. Lingkungan eksekusi TOSCA2Chef yang dibangun dalam model orkestrasi cloud mampu melakukan evaluasi terhadap dokumen TOSCA, serta mengambil informasi yang dibutuhkan dari dokumen TOSCA untuk secara otomatis diubah menjadi perintah tereksekusi dari knife pada Chef *client*. Selanjutnya knife pada Chef *client* berinteraksi dengan Chef *server* dan OpenStack dalam melakukan komputasi terhadap *platform cloud* AWS guna otomasi *deployment*, konfigurasi, dan pengelolaan sumber daya. Arsitektur lingkungan eksekusi TOSCA2Chef ditunjukkan pada gambar 4 sedangkan alur tahapan metode penelitian pada artikel ini ditunjukkan dengan diagram blok pada gambar 5. Portabilitas aplikasi *cloud multi-tier* sesuai dengan standar TOSCA yang diuji coba dan dievaluasi pada penelitian ini mengimplementasikan skenario dalam bentuk *use case* dari desain topologi layanan, seperti ditunjukkan pada gambar 6.




Gambar 4. Arsitektur lingkungan eksekusi TOSCA2Chef [14]

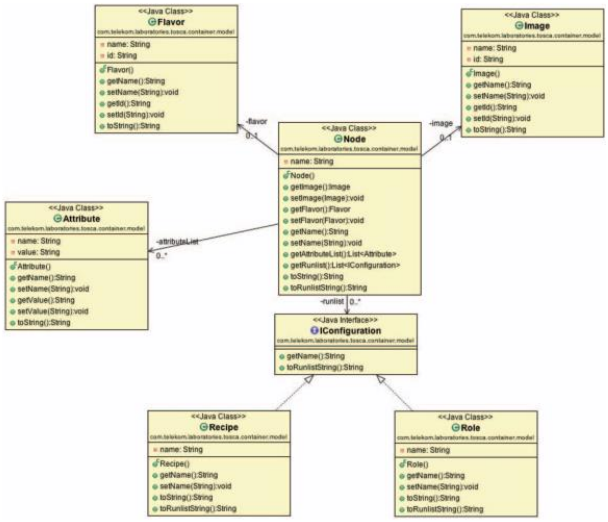



Gambar 5. Alur tahapan metode penelitian




Gambar 6. *Use case* desain topologi layanan

Lingkungan eksekusi TOSCA2Chef dibangun diatas mesin virtual OpenStack. Guna mentransformasikan dokumen TOSCA XML terparsing ke VM OpenStack yang berjalan, maka diperlukan model data *intermediary* yang merefleksikan topologi TOSCA sesuai Chef dan OpenStack. Model data *intermediary* tersebut harus mencakup domain entitas-entitas spesifik dari AWS dan OpenStack yakni infrastruktur dan topologi data. Dengan demikian guna mewujudkan portabilitas aplikasi *cloud*, mesin orkestrasi Chef dikembangkan dengan tambahan mesin eksekusi yang mengimplementasikan model data *intermediary* seperti ditunjukkan pada gambar 7. Mesin eksekusi ini dibangun sebagai bagian dari *cookbook* untuk lingkungan eksekusi TOSCA2Chef.




Gambar 7. Model data intermediary untuk lingkungan eksekusi TOSCA2Chef

## 3. HASIL DAN PEMBAHASAN

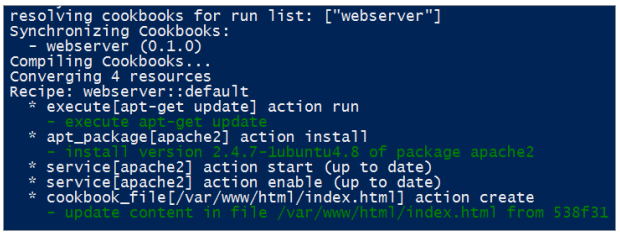
Orkestrasi dengan Chef pada artikel ini diterapkan pada dua contoh kasus. Yang pertama adalah yakni memungkinkan tim DevOps melakukan eksekusi tugas secara paralel atau tidak saling menunggu dalam *deployment* empat web services berbasis LAMP (Linux-Apache-MySQL-Python) *web service stack* secara bersamaan sebagai Software as a Service (SaaS) pada AWS, sekaligus mempermudah *system administrator* dalam mengelola *web services* tersebut. Sedangkan contoh kasus kedua yakni mewujudkan portabilitas pada aplikasi *cloud multi-tier* sesuai standar TOSCA.

### 3.1. Orkestrasi Chef dalam mewujudkan eksekusi paralel tugas tim DevOps dan otomasi tugas *system administrator*

Terkait deployment dan pengelolaan empat web services berbasis LAMP, maka masing-masing *web service* tersebut menjadi *node* dari Chef *client*. Keempat web services tersebut yakni seattle.chef.com, portland.chef.com, atlanta.chef.com, dan portland.chef.com, yang kesemuanya *deployed* pada layanan cloud

AWS. Pada dasarnya deployment *web service(s)* pada AWS terdiri atas langkah-langkah yang harus dilakukan secara sekuensial [15], namun dengan orkestrasi Chef ini maka terdapat beberapa hal yang dapat dilakukan secara simultan atau bersamaan untuk keempat node tersebut yakni konfigurasi dan pengelolaan basis data, konfigurasi dan pengelolaan *load balancer*, konfigurasi HTTPS, dan *traffic-splitting deployments* (jika diimplementasikan sebagai konfigurasi *deployment)*. Dengan dimungkinkannya beberapa langkah untuk dilakukan secara simultan maka akan menghemat waktu kerja tim DevOps dari sembilan hari kerja [16] menjadi tiga hari kerja.

Dengan adanya orkestrasi dengan Chef maka pekerjaan-pekerjaan yang biasanya dilakukan oleh *system administrator*, akan digantikan oleh Chef secara otomatis, dinamis, dan kolaboratif dengan menyesuaikan dengan kebutuhan masih-masing sistem yang diorkestrasi dan juga kebutuhan organisasi pengguna. Prosedur-prosedur yang dapat diotomasikan dicakup dalam cookbook menggunakan knife. Gambar 8 menunjukkan salah satu contoh prosedur yang dapat diotomasikan yakni dimana sub-sistem dari salah satu node melakukan *updating* dari *web server* Apache yang diotomasi oleh Chef dengan melakukan kompilasi *cookbook* yang ada. Gambar 8 menunjukkan proses *update web server* berbasis kebutuhan, yana mana biasanya tidak dapat dilakukan secara otomatis (*update* versi). Basis kebutuhan dilakukan dengan mengubah kebutuhan PHP dari salah satu *node* (*update* versi dari PHP 5.4 ke PHP 7.2.30), yang mana kebutuhan *update* versi PHP dilakukan dengan mengubah kode program dari sistem salah satu *node.*

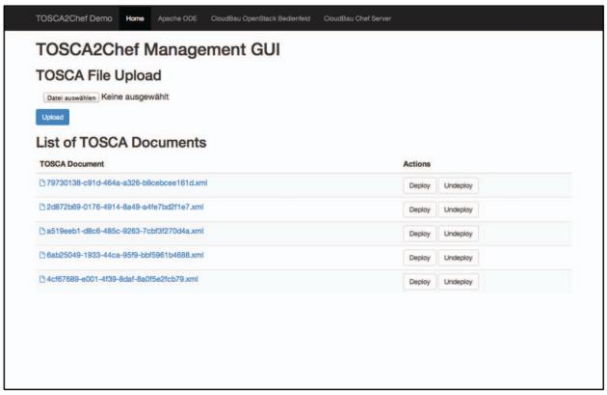



Gambar 8. Updating *web server* Apache terotomasi untuk salah satu *node*

Pendekatan DevOps pada rekayasa piranti lunak yang semakin populer dewasa ini. menentukan otomasi, orkestrasi, dan keamanan sebagai prioritas utama bagi *system administrator*. Dengan pendekatan DevOps, maka fungsi pengembang piranti lunak akan semakin mendekati fungsi system administrator. Pendekatan DevOps membutuhkan pengembang piranti lunak untuk dapat senantiasa menggelar sistem secara fungsional, sehingga tugas-tugas terotomasi yang ditentukan secara akurat oleh system administrator sehingga pengembang piranti lunak dapat mengaktifkan layanan-layanan (*services*) guna menguji kode yang telah ditulis. Sedemikian rupa, piranti lunak orkestrasi harus mengkoordinir templates dari tugas-tugas sehingga pengembang piranti lunak memiliki lingkungan layanan IT yang lengkap. Pada dasarnya pengembang piranti lunak tidak akan dapat menciptakan kode yang tepat jika pada lingkungannya hanya terdapat 75% atau bahkan 95% dari ketersediaan layanan yang bekerja (*working services*). Sedemikian rupa, profesional pada bidang keamanan juga membutuhkan layanan-layanan yang terorkestrasi dengan benar. Di masa lampau, profesional di bidang keamanan akan menunggu tersedianya staging server lengkap dengan kode yang telah dikembangkan, yang mana kemudian profesional di bidang keamanan tersebut akan melakukan *review* terhadap kode serta bagaimana kode tersebut berfungsi pada lingkungan *mocked up*. Namun dengan pendekatan DevOps dewasa ini, kode dikembangkan dan diimplementasikan langsung pada server produksi sepanjang *project* berjalan, sehingga para profesional di bidang keamanan harus menguji implementasi kode selama *project* berjalan.

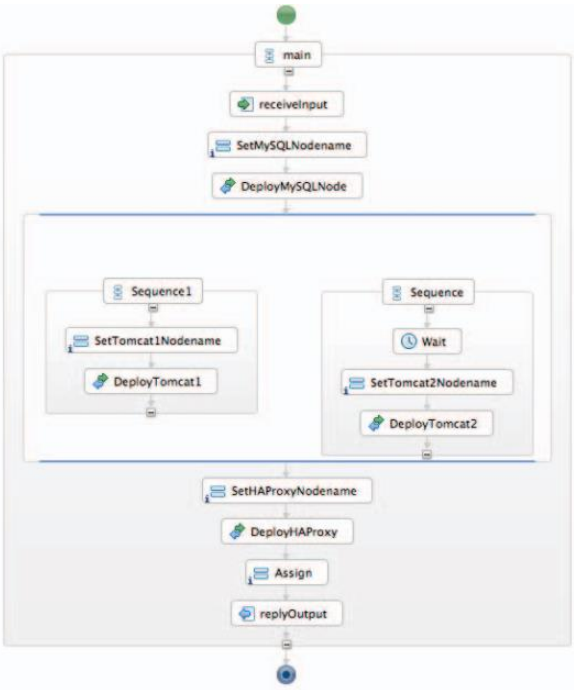
### 3.2. Evaluasi lingkungan eksekusi TOSCA2Chef

Guna kepentingan evaluasi, dikembangkan *Graphical User Interface* (GUI) yang memungkinkan antarmuka web service TOSCA2Chef dapat diakses untuk pengujian, seperti ditunjukkan pada gambar 9. GUI diimplementasikan sebagai halaman *web* guna mengunggah dokumen-dokumen TOSCA-XML sekaligus eksekusi terhadap *deployment* dan atau *un-deployment* terhadap proses-proses BPEL terkait. GUI tersebut *deployed* pada web server Apache. Dari Chef *server*, antarmuka *web service* TOSCA2Chef dipanggil guna *put* atau *get* dokumen-dokumen TOSCA. Guna proses-proses BPEL, antarmuka dari *web service* Apache ODE dipanggil guna instantiasi *deployement* dan atau *un-deployment* dari proses-proses. Antarmuka GUI Apache ODE saat deployment dari proses BPEL ditunjukkan pada gambar 10. *Deployment* dari proses BPEL akan menambah *node* sedangkan *un-deployment* akan menghapus *node*. *Node-node* yang terlibat pada evaluasi lingkungan eksekusi TOSCA2Chef merupakan keempat *node* yang telah dipaparkan sebelumnya.




Gambar 9. GUI guna eveluasi lingkungan eksekusi TOSCA2Chef

Dari proses-proses BPEL, antarmuka *web service* TOSCA2Chef dipanggil guna orkestrasi *deployment* dari topologi. Agar supaya evaluasi lingkungan eksekusi TOSCA2Chef dapat terukur secara kuantitatif (misal lama waktu yang dibutuhkan untuk *deployment)*, maka diimplementasikan skenario *use case* melalui solusi AWS CloudFormation. Basis data MySQL berikut *load balancer* Nginx guna keperluan pengukuran dipindahkan dari mesin virtual OpenStack ke AWS menggunakan perintah pada cookbook dari lingkungan eksekusi TOSCA2Chef yang ditulis dengan knife pada Chef *server*.




Gambar 10. *Deployment* proses-proses BPEL

Berdasar evaluasi terhadap lingkungan eksekusi TOSCA2Chef maka diketahui keberhasilan pengelolaan sumber daya perangkat keras secara transparan dengan instalasi OpenStack yang didukung Opscode Chef *server* sebagai mesin pengelola konfigurasi di dalam topologi. Kemudian diketahui juga bahwa *server* Apache ODE mampu melakukan eksekusi terhadap proses-proses BPEL. *Web server* Apache berfungsi sebagai *server* aplikasi yang melayani lingkungan eksekusi TOSCA2Chef, sedangkan *server* Apache ODE menjalankan proses-proses BPEL guna memicu *deployment* dan atau *un-deployment* dari topologi yang terdefinisi pada *use case*. Proses-proses BPEL melakukan eksekusi terhadap *web service* dari *web server* Apache saat setiap memicu *deployment* dan atau *un-deployment* dari topologi yang terdefinisi pada *use case*. *Web service* dari Apache *web server* sendiri melakukan eksekusi terhadap knife dari OpenStack berdasar parameter-parameter dari sistem operasi. Panggilan terhadap *web service* bersifat sinkron, dimana respon dari *web service* terkirim setelah mesin virtual telah terinisiasi dan semua tugas konfigurasi piranti lunak yang dikelola Opscode Chef telah selesai dilaksanakan. Perilaku pemblokiran ini merupakan salah satu fitur orkestrasi Chef yang menjamin terselesaikannya sekaligus keberhasilan dari *deployment* (penambahan *node*).

Terkait evaluasi pada arsitektur berikut konfigurasi yang dibangun pada penelitian ini, bahwa telah dilakukan sepuluh kali *deployment* yang bertujuan untuk menghitung rerata waktu yang dibutuhkan untuk *deployment* dari *use case* pada topologi. Setiap *deployment* dari kesepuluh *deployment* yang dilakukan adalah *deployment* terhadap keseluruhan topologi *use case* seperti yang ditunjukkan pada gambar 6, dimana dipicu oleh proses-proses *deployment*. Urutan *deployment* yang terjadi, seperti ditunjukkan pada gambar 10 yakni *deployment* pertama adalah basis data MySQL, kemudian *web server* Apache, dan yang terakhir *load balancer* Nginx. Berdasarkan pengukuran, rerata waktu yang diperlukan untuk setiap *deployment* yakni 17 menit 25 detik. Waktu yang diperlukan untuk setiap *deployment* merupakan agregasi dari *deployment* komponen tunggal, kecuali untuk dua *web server* Apache yang *deployed* secara paralel. Dengan kata lain guna mewujudkan portabilitas aplikasi *cloud multi-tier* sesuai standar TOSCA, diperlukan *deployment* semi-sekuensial dibandingkan dengan *fully paralel deployment* jika orkestrasi diterapkan guna pengelolaan *node-node* dari layanan *cloud*. *Deployment* semi-sekuensial ini diperlukan guna memastikan resep dari Opscode Chef teraplikasi dengan tepat, terutama saat tujuan dari resep konfigurasi adalah menghubungkan semua komponen yang ada. Resep akan gagal jika salah satu komponen tidak tersedia. Dengan demikian orkestrasi terhadap *deployment* yang mendukung portabilitas akan senantiasa membutuhkan proses-proses BPEL dan eksekusi pemblokiran.

Jika piranti lunak pengelolaan konfigurasi mampu untuk melakukan eksekusi tugas-tugas *inter-component* secara sekuensial setelah konfigurasi awal diaplikasikan pada topologi, maka bagian-bagian dari deployment dapat dieksekusi secara paralel. Pada penelitian ini diketahui bahwa potensi waktu *deployment* yang panjang adalah dikarenakan instalasi piranti lunak, dimana kemudian fase instalasi dan konfigurasi piranti lunak dapat dilakukan secara paralel pada aplikasi *cloud* dengan orkestrasi Chef. Pada uji coba penerapan AWS CloudFormation *use case* sebagai tindak lanjut pemenuhan standar TOSCA, diperoleh rerata waktu *deployment* sebesar 14 menit 13 detik dan rerata waktu *un-deployment* sebesar 1 menit 30 detik. Diketahui juga bahwa *load balancer* tidak berpengaruh signifikan terhadap rerata waktu *deployment* dan *un-deployment*.

## 4. PENUTUP

Berdasarkan hasil yang diperoleh diketahui bahwa layanan-layanan teknologi informasi yang mana dalam penelitian ini adalah *web server*, basis data, dan *load balancer* dapat diletakkan pada layanan *cloud* AWS dan kemudian diotomasikan secara selaras menggunakan mesin orkestrasi Chef. Orkestrasi tersebut juga melakukan pengelolaan secara otomatis terhadap *node-node* yang dilayani oleh layanan-layanan teknologi informasi tersebut yang mana merupakan berbagai macam solusi bisnis. Selain dukungan pada pendekatan DevOps, orkestrasi akan menggantikan sebagian besar fungsi dari *system administrator* dalam pengelolaan layanan-layanan teknologi informasi sacara otomatis dan selaras. Yang harus diperhatikan dalam implementasi mesin orkestrasi adalah pemahaman terhadap cara kerja dan cara melakukan koordinasi dengan penyedia layanan *cloud* untuk mesin orkestrasi yang akan digunakan. Hal ini dikarenakan berbeda mesin orkestrasi, berbeda dalam cara kerja dan bagaimana untuk melakukan koordinasi dengan penyedia layanan *cloud*. Diperlukan penelitian tentang orkestrasi terhadap otomasi layanan-layanan IT pada lingkungan *on premise*. Hal ini perlu dilakukan dikarenakan lingkungan *on premise* masih sangat banyak digunakan di Indonesia terutama di lingkungan industri. Perlu dilakukan juga penelitian tentang mesin orkestrasi yang paling cocok untuk lingkungan dan kebutuhan spesifik.

Kapabilitas *application packaging* dalam *clouds* yang memungkinkan portabilitas dan *re-usability* merupakan pra-kondisi yang penting dalam mewujudkan manfaat dari layanan *cloud* tervirtualisasi. Hal tersebut membutuhkan standar yang pasti dan terdefinisi dengan baik dimana diadopsi oleh industri. Standar-standar tersebut haruslah memungkinkan aplikasi-aplikasi untuk dapat digunakan kembali, diubah konfigurasinya sesuai parameter-parameter yang dibutuhkan, *portable*, *interoperable* antar *platform*, serta memungkinkan integrasi antara piranti lunak dengan infrastruktur terutama guna Platform as a Service (PaaS) dan Software as a Service (SaaS).

Artikel ini mengacu pada TOSCA guna menekankan pada fitur-fitur portabilitas dari lingkungan cloud, dengan membangun *use case model* dari aplikasi berbasis *web* sesuai spesifikasi TOSCA. Telah berhasil dibangun juga lingkungan eksekusi TOSCA2Chef yang memungkinkan *deployment* dari *use case* pada infrastruktur *cloud* yakni AWS. Hingga saat ini, TOSCA belum mendefinisikan lingkungan eksekusi sebagai bagian dari standar portabilitas sehingga dalam penelitian ini ditunjukkan bahwa implementasi TOSCA memungkinkan portabilitas dimana masih terdapat keterbatasan. Untuk kedepannya diharapkan bahwa lingkungan eksekusi juga terdefinisi dalam TOSCA sehingga memungkinkan setiap operator *cloud* mengadopsi standar dan menawarkan antarmuka yang diharapkan guna *doployment* aplikasi *cloud* yang dimodelkan dengan TOSCA. TOSCA juga belum mendefinisikan struktur dan penamaan entitas tipe *node*, sehingga definisi dari sumber daya *cloud* yang ada seperti *images* dan konfigurasi mesin virtual, serta konfigurasi piranti lunak diserahkan kepada masing-masing operator *cloud*. Agar supaya dapat keluar dari batasan portabilitas yang dimungkinkan saat ini, diharapkan adanya standar yang disepakati antar operator

*cloud* tentang entitas-entitas yang menjadi komponen dari infrastruktur *cloud*. Spesifikasi TOSCA juga hingga saat ini belum mendukung aplikasi *federated cloud* sehingga akan menjadi subject penelitian yang melanjutkan penelitian pada artikel ini.

## DAFTAR PUSTAKA

[1] L. Lin and P. Lin, "Orchestration in web services and real-time communications," *IEEE Commun. Mag.*, 2007.
[2] D. Jeya Mala and A. Pradeep Reynold, "Towards Green Software Testing in Agile and DevOps Using Cloud Virtualization for Environmental Protection," 2020.
[3] T. Binz, G. Breiter, F. Leyman, and T. Spatzier, "Portable cloud services using TOSCA," *IEEE Internet Comput.*, vol. 16, no. 3, pp. 80–85, 2012.
[4] F. LEYMANN, C. FEHLING, R. MIETZNER, A. NOWAK, and S. DUSTDAR, "MOVING APPLICATIONS TO THE CLOUD: AN APPROACH BASED ON APPLICATION MODEL ENRICHMENT," *Int. J. Coop. Inf. Syst.*, vol. 20, no. 03, pp. 307–356, Sep. 2011.
[5] A. W. Maia and P. P. M. Farias, "Transactions as a Service," 2019, pp. 415–423.
[6] S. Strauch, V. Andrikopoulos, T. Bachmann, and F. Leymann, "Migrating application data to the Cloud using Cloud data patterns," in *CLOSER 2013 - Proceedings of the 3rd International Conference on Cloud Computing and Services Science*, 2013, pp. 36–46.
[7] E. Bauer, "Cloud Automation and Economic Efficiency," *IEEE Cloud Comput.*, 2018.
[8] J. Stanger, "Make Your IT Automation Systems Play Together Like a Symphony Cloud Orchestration with Chef," *ADMIN Network & Security*, 2017. [Online]. Available: https://www.admin-magazine.com/Archive/2017/42/Make-Your-IT-Automation-Systems-Play-Together-Like-a-Symphony. [Accessed: 13-Mar-2020].
[9] C. S. Draft and P. R. Draft, *Topology and Orchestration Specification for Cloud Applications Committee Specification Draft 06 /*, no. November. 2012, pp. 1–114.
[10] IBM, "IBM SmartCloud Enterprise (SCE)." 2014.
[11] S. Management, S. Purchasing, P. Must, and M. J. Turner, "Orchestration Simplifies and Streamlines Virtual and Cloud Data Center Management Industry Trends : Dynamic , Heterogeneous Data Centers Demand," 2013.
[12] OASIS, *Web Services Business Process Execution Language*. 2009, pp. 3520–3520.
[13] T. A. S. Foundation, "{A}pache {ODE} ({O}rchestration {D}irector {E}ngine)---{W}ebsite." 2018.
[14] G. Katsaros, M. Menzel, A. Lenk, J. R. Revelant, R. Skipp, and J. Eberhardt, "Cloud Application Portability with TOSCA, Chef and Openstack," in *2014 IEEE International Conference on Cloud Engineering*, 2014, pp. 295–302.
[15] Amazon Web Services, "AWS Elastic Beanstalk," *Date Accessed: 2020.05.01*. 2010.
[16] J. Díaz, R. Almaraz, J. Pérez, and J. Garbajosa, "DevOps in practice," in *Proceedings of the 19th International Conference on Agile Software Development Companion - XP '18*, 2018, pp. 1–3.