BAB II
LANDASAN TEORI

Pada bab ini, peneliti akan menguraikan teori-teori yang mendukung dalam implementasi *Automation Deployment*. Berikut akan dijelaskan teori-teori tersebut dengan lebih lengkap.

2.1. DevOps (*Development and Operations*)

2.1.1 Pengertian DevOps

DevOps merupakan pendekatan kolaboratif antara bagian pengembangan aplikasi (*dev*) dan bagian operasi aplikasi (*ops*). DevOps juga merupakan sebuah *culture* yang ditujukan untuk pembangunan perusahaan yang berkelanjutan [2]. DevOps adalah serangkaian praktik yang mengautomasikan proses antara pengembangan aplikasi dan tim pengembang agar mereka dapat melakukan proses *build*, *test* dan *release* perangkat lunak lebih cepat dan lebih handal.

2.1.2 Nilai inti dari DevOps

Nilai inti DevOps biasanya dijelaskan dengan singkatan CAMS. Merupakan singkatan dari *Culture*, *Automation*, *Measurement*, dan *Sharing* [3].

DevOps memecah hambatan antara tim, menempatkan fokus pada orang dan interaksi mereka di atas proses dan alat. Kejujuran, keterbukaan, dan ketulusan adalah bagian paling berharga dari tim DevOps. Disarankan untuk mengajukan pertanyaan dan berbagi pengetahuan, pelajaran, dan penemuan. Perilaku yang membantu mengoptimalkan proses dan memahami mengapa proses yang ada gagal, didukung, dan dihargai. Mempromosikan lingkungan yang aman untuk inovasi dan produktivitas adalah tantangan utama bagi manajemen perusahaan [4].

Automasi membawa sistem dalam urutan. Ini membantu untuk secara cerdas menggambarkan perubahan yang dibuat terhadap lingkungan. Untuk memahami apa yang harus diubah dalam produk, tim menggunakan log, statistik, dan *feedback* yang dikumpulkan secara konstan. Analisis proses kerja terus dilakukan, setiap perubahan ditinjau untuk memahami apakah aplikasi menjadi lebih baik dan apakah arah yang dipilih sudah benar [4].

2.1.3 Manfaat DevOps

Perusahaan yang mengadopsi praktik DevOps mendapatkan banyak keuntungan teknis dan organisasi. Beberapa diantaranya adalah sebagai berikut:

- **Meningkatkan stabilitas dan kualitas**. Ketika semua anggota tim memiliki lingkungan pengembangan yang sama, integrasi pengujian berkelanjutan dapat dipercepat. Pengujian berkelanjutan ini memungkinkan siklus rilis lebih cepat dan lebih sering dan untuk masalah dan kegagalan yang lebih mudah diidentifikasi [5].

- **Meningkatkan efektivitas organisasi**. Lebih banyak waktu dihabiskan untuk memingkatkan nilai dan kualitas produk [6].

- **Meningkatkan pengalaman pelanggan**. Kemampuan untuk menerima *feedback* terus menerus dan lebih cepat memperkenalkannya ke dalam pengembangan proyek mengarah pada peningkatan pendapatan dan peningkatan kepuasan pelanggan [7].

- **Reaksi cepat terhadap perubahan pasar dan permintaan pelanggan**. Kemampuan untuk mempertahankan tingkat penyebaran yang tinggi ditransformasikan menjadi nilai bisnis dalam dua cara utama: seberapa cepat suatu organisasi dapat berpindah dari ide ke sesuatu yang dapat ditransfer ke pelanggan dan berapa banyak percobaan yang dapat dilakukan organisasi secara bersamaan. Tingkat penyebaran yang tinggi

STT-NF

memungkinkan untuk melakukan percobaan dengan cepat dan terus menerus [8].

- **Automation**. Dengan mengganti prosedur manual dengan automasi membuat manajemen infrastruktur lebih efisien. Hal ini meningkatkan tingkat penggunaan dan kemudahan anda mengelola semua sumber daya, baik *on premises*, pada *cloud*, atau di lingkungan *hybrid*. Respon cepat terhadap perubahan kebutuhan pada bisnis sangat penting. Automasi memungkinkan anda meningkatkan atau menurunkan respon terhadap permintaan. Automasi memastikan konfigurasi konsisten di seluruh jaringan anda, sehingga setiap pengembang memiliki lingkungan yang sama dan pengembang baru dapat dengan mudah menggunakan lingkungan kerja dan mulai bekerja. Konsistensi memberi anda lebih banyak kontrol, dan kontrol mengurangi resiko. Manfaat nyatanya adalah anda memiliki proses standar untuk menyediakan *server* [9].

- **Tingkat komunikasi yang tinggi antara departemen dan anggota tim** [10].

- **Mengurangi risiko perubahan**. DevOps mengurangi risiko perubahan dengan membuat banyak perubahan kecil dan *incremental* alih-alih lebih sedikit. Perubahan kecil lebih mudah untuk ditinjau dan diuji. Karena cakupan setiap perubahan kecil dan umumnya terisolasi, maka jauh lebih mudah untuk memperbaiki kesalahan yang mungkin terjadi [11].

2.1.4 Praktek pada DevOps

- *Continuous Integration* (CI)

  CI adalah praktik pengembangan yang mengharuskan pengembang untuk mengintegrasikan kode ke dalam *mainline* sesering mungkin, dan setiap *check-in* kemudian diverifikasi oleh *automated build* yang mengkompilasi kode dan menjalankan rangkaian uji otomatis terhadapnya, memungkinkan tim mendeteksi masalah sejak dini.

- *Continuous Delivery* (CD)

  CD adalah praktek *software development* dimana para pengembang yang melakukan perubahan pada code, sudah melakukan *build* & *test* yang dijalankan otomatis oleh CI dan siap untuk *deploy* ke *environment production*.

- *Continuous Deployment*

  Continuous Deployment merupakan salah satu rangkaian setelah CI dan CD selesai dijalankan. Umumnya organisasi/ perusahaan memiliki e*nvironment test/ development*, dan disinilah fungsi utama *Continuous Deployment*, yaitu ketika hasil dari *Continuous Integration* sudah dinyatakan baik, tim pengembang dapat segera melihat perubahan pada *environment test/ development/ production*.

- *Configuration Management*

  Configuration Management adalah praktek dalam proses *System Engineering* yang memiliki tujuan untuk me-*maintain* konfigurasi sebuah produk, dan memastikan konsitensinya dalam seluruh *environment*. Dengan menggunakan *Configuration Management*, proses konfigurasi produk dapat diotomatisasi, distandardisasi dan mengurangi proses konfigurasi yang manual. Tahap selanjutnya, *Configuration Management* akan mempermudah dalam konfigurasi banyak *server* dan dapat meminimalisir kesalahan, karena konfigurasi ditulis dalam *code*, tidak lagi menjalankan perintah manual.

- *Infrastructure as a Code* (IaaC)

  Infrastructure as a Code adalah sebuah praktek dalam *System Architecture* yang mana infrastruktur sebuah produk didefinisikan dalam *code* yang dapat diprogram, distandardiasikan dan mudah untuk diduplikasi. Produk skala menengah, mungkin membutuhkan lebih dari satu mesin. Dengan IaaC, tim pengembang dapat dengan mudah menambah mesin melalui satu baris kode.

STT - NF

- *Monitoring*

  Sebuah produk haruslah di-*monitoring* untuk mengetahui bagaimana produk digunakan oleh pengguna. Dalam praktek DevOps, *monitoring* merupakan hal yang sangat penting. Tim pengembang harus mengetahui bagaimana perubahan kodenya berdampak pada produk juga penggunanya melalui *monitoring tools*.

- *Logging*

  Log aplikasi adalah salah satu cara untuk mengetahui apakah produk kita berjalan dengan baik atau tidak. Namun seiring dengan tingkat kompleksitas sebuah produk, ada banyak *log* komponen yang harus diterima dan dianalisis. Dan *log* tersebut haruslah terpusat, tidak terpisah-pisah.

- *Communication* & *Collaboration*

  Salah satu aspek utama dalam praktek DevOps yaitu meningkatnya komunikasi dan kolaborasi dalam sebuah organisasi/ perusahaan, baik dalam bentuk fisik maupun non fisik. Praktek DevOps yang berjalan dengan baik, akan meningkatkan aspek komunikasi dan kolaborasi tidak hanya pada tim pengembang, namun juga tim *marketing*, *sales*, *operations*, dan tim lain yang ada didalam organisasi/ perusahaan

Masalah utama yang perlu diselesaikan DevOps adalah lambatnya proses *delivery*. tujuan utama DevOps adalah mengurangi biaya proses pengembangan dengan mengautomasi dan mengintegrasikan seluruh sistem yang memungkinkan *developer* lebih fokus ke proses *development* [12]

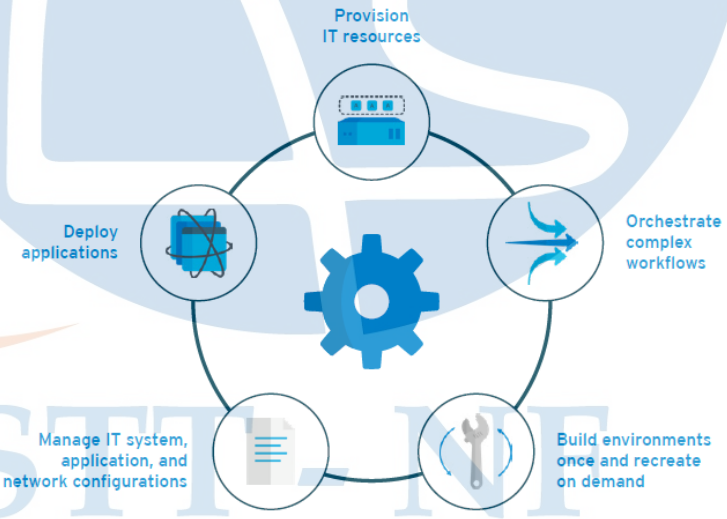
2.2. Pengertian *Automation Deployment*

Tahap Penyebaran (*Deployment*) adalah tahap dimana sistem dibuat tersedia bagi komunitas pengguna. Proses penyebaran harus direncanakan dengan baik sehingga meminimalkan *downtime* dan dampak untuk mengakhiri produktivitas

pengguna. Hal ini tidak hanya mencakup perangkat keras dan perangkat lunak tetapi pengguna akhir.

Automation Deployment memungkinkan aplikasi untuk digunakan di berbagai lingkungan yang digunakan dalam proses pengembangan, serta lingkungan produksi. Teknik ini menghasilkan penerapan yang lebih efisien, andal, dan dapat diprediksi. Proses *Automation Deployment* meningkatkan produktifitas tim Dev dan Ops dan memungkinkan mereka untuk mendeploy lebih cepat dan membangun *software* yang lebih baik untuk *end-user* [13]. *Tools* yang dapat digunakan untuk menerapkan *Automation* salah satunya adalah Ansible.

Automasi adalah satu-satunya cara untuk dapat berhasilnya mengelola lingkungan IT dalam jangka panjang. Menciptakan sebuah pendekatan *enterprise-wide* memungkinkan kita mengautoamsi tidak hanya proses IT, tetapi juga seluruh teknologi dan organisasi, pengembangan proses DevOps dapat dilihat pada gambar 2.1 di bawah ini.



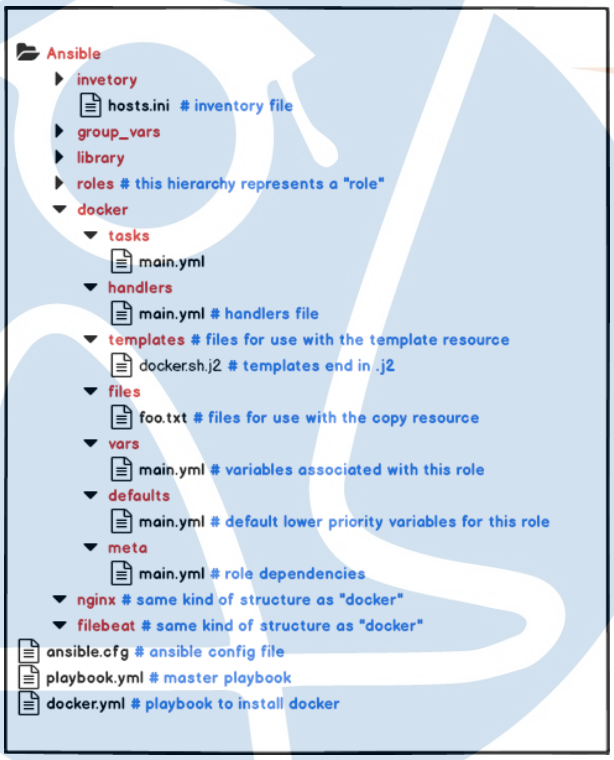
Gambar 2.1: *Automate IT Process*

2.3. Ansible

Ansible adalah mesin autoamsi *open source* yang mengautomasi penyediaan perangkat lunak, Manajemen Konfigurasi, dan pemasangan aplikasi [1]. Ansible disertakan sebagai sebuah *provisioning tool* yang dikembangkan oleh RedHat.

Dengan kata lain *software* yang dapat membantu seorang DevOps untuk melakukan automasi di *server*nya. Ansible ditulis dalam satu bahasa sederhana yang disebut YAML.

Berbeda dengan Pupet dan Chef, Ansible bersifat *agent-less*, yang artinya kita tidak perlu memiliki agent tambahan yang terpasang sebelumnya karena Ansbile bekerja cukup pada koneksi ssh. Struktur dasar pada Ansible dapat dilihat pada gambar 2.2



Gambar 2.2: Struktur Ansible

2.3.1. Terminologi pada Ansible

a. Controller Machine

Mesin dimana Ansible diinstalasi dan bertanggung jawab untuk menjalankan *provisioning* pada *server* yang dikelola.

b. Ansible *Playbook*

Titik masuk untuk Ansible provisioning, dimana automasi didefinisikan melalui tugas (*tasks*) menggunakan format YAML.

c. Task

Blok yang mendefinisikan satu prosedur untuk dieksekusi, sebagai contoh instalasi *package* tertentu.

d. Module

Module merupakan abstraksi dari tugas sistem, seperti berkaitan dengan *package* atau membuat dan mengubah *file*. Ansible memiliki banyak modul *built-in*, namun dapat juga dibuat modul *custom*.

e. Role

Cara yang telah ditentukan sebelumnya untuk mengatur *Playbook* dan *file* lainnya untuk memfasilitasi berbagi pakai dan menggunakan kembali bagian dari *provisioning*.

f. Play

Provisioning yang dieksekusi mulai dari awal sampai akhir disebut dengan *play*. Dengan kata lain, eksekusi dari *Playbook* disebut dengan *play*.

g. Facts

Variable global yang memuat informasi tentang sistem, seperti *interface* jaringan atau sistem operasi.

h. Handlers

Handlers digunakan untuk memicu perubahan status dari *service*, seperti me-restart atau menghentikan *service*.

i. Inventory

Pada Ansible dikenal satu istilah dengan nama "*inventory*", *inventory* yang dimaksudkan di sini adalah *inventory server* kita. Yaitu sebuah file berisikan daftar *server* yang kita akan konfigurasi menggunakan Ansible. *Server* ini merupakan target *deployment* atau apapun yang kita lakukan dengan Ansible yang ada di komputer.

j. Ad-Hoc Commands

Perintah yang dapat diketik pada terminal untuk melakukan sesuatu yang sangat cepat. *Ad-hoc commands* adalah fitur untuk mulai memahami dasar-dasar tentang apa yang dapat dilakukan oleh Ansible sebelum mempelajari sebuah *Playbook*. Seorang dapat menjalankan satu baris *command* cepat di Ansible tanpa harus menulis *Playbook.*

k. Bahasa pemrograman YAML

Bahasa pemrograman yang digunakan untuk membuat sebuah *Playbook*. YAML (dibaca yamel) adalah sebuah format serialisasi data yang mudah dibaca manusia secara umum yang mengambil konsep bahasa pemrograman C, Perl, Python dan ide lain dari format XML dan format email. Ekstensi file YAML adalah **.yaml** atau **.yml**. Berikut pedoman dalam membuat file yaml:

- YAML tidak menggunakan tab, tetapi menggunakan spasi (indentasi)
- YAML sensitif terhadap huruf besar dan kecil
- Komentar dimulai dengan tanda pagar (#), dapat dimulai di mana saja pada suatu baris dan berlanjut hingga akhir baris.
- Daftar anggota dilambangkan dengan tanda hubung (-) dengan satu anggota per baris.
- Batasan blok dibatasi dengan tanda (|)
- Beberapa dokumen dalam satu aliran dipisahkan oleh tiga tanda hubung (---)

2.3.2. Perbandingan *Tools Automation*

Selain Ansible, Chef, Puppet, dan SaltStack adalah alat untuk Manajemen Konfigurasi, masing-masing menyajikan jalur yang berbeda untuk mencapai tujuan bersama mengelola infrastruktur *server* berskala besar secara efisien. Keempat alat Manajemen Konfigurasi tersebut dirancang untuk mengurangi kerumitan mengkonfigurasi sumber daya infrastruktur terdistribusi. Berikut ini merupakan tabel perbandingan *tools automation* tersebut:

Metrics	Chef	Puppet	Ansible	SaltStack
Availability	✓	✓	✓	✓
Easy of Setup	Not very easy	Not very easy	Easy	Not very easy
Management	Not very easy	Not very easy	Easy	Easy
Scalability	Highly Scalable	Highly Scalable	Highly Scalable	Highly Scalable
Configuration Language	DSL(Ruby)	DSL(Puppet)	YAML(Python)	YAML(Python)
Interoperability	High	High	High	High
Pricing (up to 100 *nodes*)	$13700	$11200-$19900	$10,000	$15,000(approx.)

Tabel 2.1: Perbandingan Tools Automation

Sebagai pendatang baru dibandingkan dengan Puppet, Chef dan SaltStack, Ansible dikembangkan untuk menyederhanakan tugas orkestrasi dan Manajemen Konfigurasi yang rumit. *Platform* ini ditulis dengan *Python* dan memungkinkan pengguna untuk melakukan skrip perintah di YAML sebagai paradigma pemrograman *imperatif*. Ansible menawarkan beberapa model *push* untuk mengirim modul perintah ke *node* melalui *ssh* yang dieksekusi secara berurutan. Kelebihan Ansible adalah tidak memerlukan agen tambahan di setiap sistem, dan modul dapat berada di *server* apa pun.

Setiap *tools* ditujukan untuk segmen pengguna yang berbeda dalam target pasar yang sama. Tim DevOps yang berinvestasi dalam solusi Manajemen Konfigurasi harus mempertimbangkan persyaratan khusus di sekitar alur kerja mereka agar tepat sasaran. Untuk memilih solusi Manajemen Konfigurasi yang tepat yang sesuai dengan organisasi, maka perlu pertimbangan *architecture*, *operational model*, *features*, dan *usability and support*, di antara aspek tehnikal dan bisnis lainnya.

2.4. SSH (*Secure Shell*)

Protokol SSH (*Secure Shell*) adalah metode untuk melakukan login jarak jauh dari satu komputer ke komputer lainnya [14]. SSH memanfaatkan kriptografi untuk melakukan komunikasi data pada perangkat jaringan agar lebih aman. Dalam konsepnya penggunaan SSH ini harus didukung oleh *server* maupun perangkat atau komputer klien yang melakukan pertukaran data. Keduanya harus memiliki SSH *Server* dari sisi komputer *server* dan SSH Client untuk komputer penerima.

Banyak digunakan pada sistem operasi berbasis Linux dan Unix untuk mengakses akun Shell, SSH dirancang sebagai pengganti Telnet dan shell remote tidak aman lainnya, yang mengirim informasi, terutama kata sandi, dalam bentuk teks sederhana yang membuatnya mudah untuk dicegat. Enkripsi yang digunakan oleh SSH menyediakan kerahasiaan dan integritas data melalui jaringan yang tidak aman seperti internet. Ansible sendiri bekerja di koneksi SSH remote ke *host* yang ingin di deploy atau dilakukan automasi tersebut.

2.5. *Server*

Server adalah komputer yang menyediakan jenis layanan tertentu dalam sebuah jaringan komputer [15]. Aplikasi yang disimpan di komputer ini dan terminal komputer lain terhubung, dapat mengaksesnya. Komputer *server* ini memberikan pelayanan bagi sejumlah komputer yang saling berhubungan, dalam melakukan akses data misalnya untuk pembatasan akses, melakukan kontrol data, dan aliran data yang terjadi. *Server* mempunyai banyak fungsi, beberapa diantaranya adalah, menyimpan *database* aplikasi, memberikan keamanan dengan firewall. Kesimpulannya *server* adalah sebuah komputer yang berperan untuk melayani, mengelola, serta mengatur semua keperluan komputer client dalam sebuah jaringan komputer.

Tugas yang dilakukan oleh komputer *server* tergantung pada penggunaan komputer *server*. Fungsi *server* secara umum dilakukan oleh sebuah *server* komputer adalah:

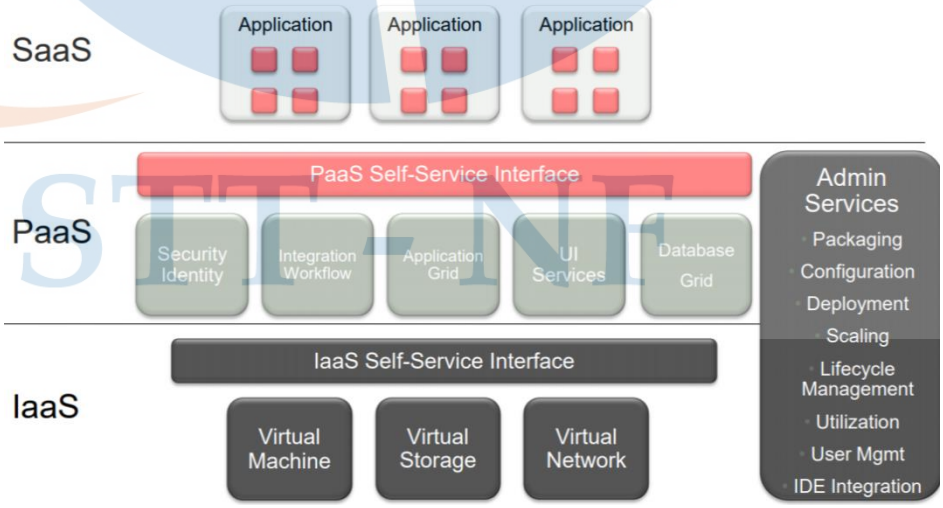
- Menyimpan aplikasi dan database yang dibutuhkan oleh komputer yang terhubung.

- Menyediakan fitur keamanan komputer
- Melindungi semua komputer terhubung menggunakan firewall
- Menyediakan IP address untuk mesin komputer terhubung.

2.6. Infrastructure as a Service (IaaS)

IaaS menyediakan abstrak dari sumber daya infrastruktur IT seperti penyimpanan dan memori sebagai layanan. Penyedia layanan *cloud* mengelola infrastruktur fisik, provisi infrastruktur dari sistem operasi [16].

Cloud Server merupakan model layanan *cloud* berbasis IaaS, yang biasa disebut juga sebagai *virtual private server*. terdapat dua tipe *cloud server*: *logical* dan *physical*. sebuah *cloud server* disebut *logical* apabila dibangun, dihosting dan di *delivery* melalui platform *cloud computing*. Sedangkan *cloud server physical* tidak dibagikan atau didistribusikan. Ini umumnya dikenal sebagai *cloud server* khusus. Berbeda dengan layanan SaaS dan PaaS, IaaS menawarkan sumber daya seperti mesin virtual (VM – *Virtual Machine*), *disk image library*, blok penyimpanan, objek penyimpanan, *firewall*, *load balancer*, alamat IP, dan *Virtual Local Area Netwrok* (VLAN). Perbedaan antara ketiga infrastruktur tersebut dapat dilihat pada gambar 2.3 di bawah ini.



Gambar 2.3: *Cloud Computing Service Models*

Pada penelitian ini, peneliti menggunakan *cloud server* untuk melakukan uji coba dan simulasi *automation deployment*, hal ini dilakukan secara *virtual* juga untuk mengenalkan *deployment* pada sisi *cloud environment*.

2.7. Web Application

Seperti banyak *website* pada umumnya, *Web Application* bersifat dinamis dan terus berkembang, dibuat menggunakan HTML, CSS, dan JavaScript. Selain itu juga menggunakan bahasa pemrograman web seperti PHP, Ruby, atau Python. Sementara untuk kebutuhan yang lebih kompleks bisa menggunakan *framework*, termasuk juga *package* pendukungnya seperti *web server*, *database engine*, dan SSL sebagai protokol keamanannya.

2.7.1. Laravel

Laravel adalah sebuah *framework* PHP yang dirilis dibawah lisensi MIT, dibangun dengan konsep MVC (model view controller) ditulis dalam PHP yang dirancang untuk meningkatkan kualitas perangkat lunak dengan mengurangi biaya pengembangan awal dan biaya pemeliharaan. Laravel, pada intinya, tentang melengkapi dan memungkinkan pengembang. Sasarannya adalah memberikan kode dan fitur yang jelas dan sederhana yang membantu pengembang belajar, memulai, dan mengembangkan dengan cepat, dan menulis kode yang sederhana, jelas, dan akan bertahan lama [17]. Pada kasus ini, peneliti melakukan automasi untuk mendeploy Laravel ke *server* agar penyebaran yang dilakukan antara Dev dan Ops lebih mudah.

STT - NF

2.7.2. Nginx (Engine x)

Nginx (*engine x*) adalah *server* proxy HTTP dan reverse, *server* mail proxy, dan *server* proxy TCP/UDP yang ditulis oleh Igor Sysoev [18].

Laravel dapat berjalan dengan dukungan *web server*. Peneliti menggunakan Nginx sebagai web *server* karena mudah di install, juga konsumsi sumberdaya yang tidak boros.

2.7.3. MySQL

MySQL merupakan software open source database yang paling populer di dunia, dimana saat ini digunakan lebih dari 100 juta pengguna di seluruh dunia. Dengan kehandalan, kecepatan dan kemudahan penggunaannya, MySQL menjadi pilihan utama bagi banyak pengembang software dan aplikasi baik di platform web maupun desktop. Pengguna MySQL tidak hanya sebatas pengguna perseorangan maupun perusahaan kecil, namun perusahaan seperti Yahoo!, Alcatel-Lucent, Google, Nokia, Youtube, Wordpress dan Facebook juga merupakan pengguna MySQL [19].

2.7.4. SSL (Secure Socket Layer)

Secure Sockets Layer (SSL) adalah teknologi komunikasi yang mengenkripsi saluran komunikasi antara situs web di *server* web dan browser di komputer. Browser terhubung ke situs web aman dengan SSL melalui protokol HTTPS yang beroperasi pada port 443. Dalam infrastruktur yang aman, Certificates memainkan peran penting dalam mengenkripsi semua data yang dikirimkan. Certificates digunakan bersama oleh situs web dan browser dalam menegosiasikan sesi aman antara komunikasi browser-ke-*server* atau *server*-ke-*server* [20]. Dengan kata lain SSL merupakan protokol yang menyediakan saluran aman antara dua mesin, dan fasilitas untuk melindungi data dan mengidentifikasi mesin yang dikomunikasikan.

2.8. Penelitian Terkait

Dalam penelitian ini peneliti melakukan studi literatur penelitian terkait sebagai komparasi dan keterkaitan dengan masalah yang peneliti ambil. Hal ini bertujuan untuk mengetahui posisi penelitian yang dilakukan. Berdasarkan tabel 1 peneliti menyimpulkan bahwa penelitian Nishant Kumar Singh, Sanjeev Thakur, Himanshu Chaurasiya, dan Himanshu Nagdev yang berjudul *Automated Provisioning of Application in IAAS Cloud using Ansible Configuration Management*, penelitian tersebut membahas mengenai permasalahan yang sama

yaitu automasi untuk proses *deployment* aplikasi yang juga menggunakan *tools* Ansible. Perbedaan yang mendasar diantaranya Infrastruktur yang digunakan pada penelitian tersebut menggunakan AWS EC2 yang digunakan khusus untuk develop dan deploy aplikasi. Berbeda dengan penelitian No. 1 dan No. 3 yang menggunakan *tools WebLogic Scripting Tool* (WLST) dan SaltStack, Ansible hanya bekerja cukup dengan koneksi ssh. Kesimpulan dari penelitian-penelitian dibawah ini dapat ditentukan satu solusi yaitu penerapan *Automation Deployment* adalah jawaban atas permasalahan pengelolaan *server* pada lingkungan yang kompleks. Daftar penelitian terkait yang peneliti temukan bisa dilihat di tabel dibawah ini.

No	Judul Penelitian	Tahun	Kesimpulan
1	AUTO DEPLOYMENT APPLICATION FILE & AUTO MAILALERT SERVER STATUS EXECUTOR TOOLS FOR COMPASS PROJECT IN PT ABC Oleh: Sasmito Budi Utomo dan Novita Kumala Sari, Program Studi Manajemen Informatika, Politeknik Manufaktur Astra, Jakarta, Indonesia.	2015	Penelitian pada proyek COMPASS PT. ABC yang membahas tentang *auto deployment* dan *mail alert* menggunakan *WebLogic Scripting Tool* (WLST) dengan metode *Throwaway Prototyping*. Lebih fokus mengenai permasalahan automasi untuk proses *deployment* dan *monitoring* status *server* secara *runtime*. Sekaligus melakukan pengujian dengan membandingkan jalannya proses manual dan automasi.
2	AUTOMATED PROVISIONING OF APPLICATION IN IAAS CLOUD USING ANSIBLE	2015	penerapan *Infrastruktur as a Service* (IaaS) dalam penelitian ini menggunakan *Ansible* adalah solusi untuk menciptakan lingkungan automasi dalam *cloud*.

	CONFIGURATION MANAGEMENT Oleh: Nishant Kumar Singh, Sanjeev Thakur, Himanshu Chaurasiya, dan Himanshu Nagdev		Automasi dilakukan untuk melakukan *deployment* pada aplikasi CRM (*Customer Relationship Management*) proses instalasi dan konfigurasi menggunakan *Ansible* untuk instalasi aplikasi dan *configuration management*, dan AWS (*Amazon Web Service)* EC2 sebagai IaaS.
3	A SYSTEM FOR APPLICATION DEPLOYMENT AUTOMATION ON CLOUD ENVIRONMENT Oleh: Sagar Narendrasing Deshmukh dan Mr. H. P. Khandagale, Computer Science and Technology Department of Technology Shivaji University, Kolhapur, India.	2017	Penelitian ini berisi tentang metode inovatif untuk menginstall dan mengkonfigurasi aplikasi SugarCRM di *environment* yang berbeda. Menggunakan *tools SaltStack* untuk melakukan *deployment application* pada *cloud Amazon Web Service*. Rancangan berfokus untuk mengurangi waktu proses *deployment* dan membantu mengurangi biaya pengembangan.

Tabel 2.2: Penelitian Terkait