BAB II
TINJAUAN PUSTAKA

2.1 Landasan Teori

2.1.1 Manajemen Infrastruktur TI

Manajemen merupakan proses perencanaan, pengorganisasian, pengarahan, dan pengendalian seluruh sumber daya yang dimiliki untuk mencapai tujuan organisasi. Manajemen Infrastruktur TI adalah proses pengelolaan seluruh komponen TI baik dari segi operasional maupun layanan dalam sebuah organisasi. Infrastruktur TI meliputi berbagai perangkat keras, perangkat lunak *system* dan aplikasi, jaringan untuk komunikasi data, data center, serta layanan berbasis cloud. Dalam konteks teknologi informasi (TI) manajemen memiliki peran penting untuk mengatur sumber daya TI seperti hardware, software, data, serta network *service* agar sejalan dengan kebutuhan bisnis dan strategi organisasi. Hal ini bertujuan untuk memastikan layanan tersedia, handal, dan terlindungi dari gangguan. Terdapat empat aspek utama manajemen infrastruktur TI yaitu *reliability & availability, security, scalability & capacity, continuity & disaster recovery*.

Reliability & *availability* memastikan layanan dapat diakses tanpa gangguan dengan menggunakan teknologi seperti high-availability *cluster*ing, failover otomatis, dan monitoring secara real time, penelitian oleh Somasekaram dkk., (2022) dengan judul "High-Availability *Cluster*s: A Taxonomy, Survey, and Future Directions" mengemukakan bahwa penerapan *high-availability clusters* dapat meningkatkan ketersediaan aplikasi secara signifikan. *Security* menerapkan proteksi system dari ancaman serangan siber melalui berbagai teknologi seperti *zero trust network, intrusion detection*, dan berbagai *tools* lainnya. Penelitian oleh Sarkar dkk., (2022) dengan judul "Security of Zero Trust Networks in Cloud Computing: A Comparative Review" mengimplementasikan teknologi zero trust untuk keamanan *system cloud computing. Scalability & Capacity* memastikan bahwa system dapat menangani lonjakan permintaan tanpa gangguan kinerja. Dan Continuity & Disaster Recovery memastikan

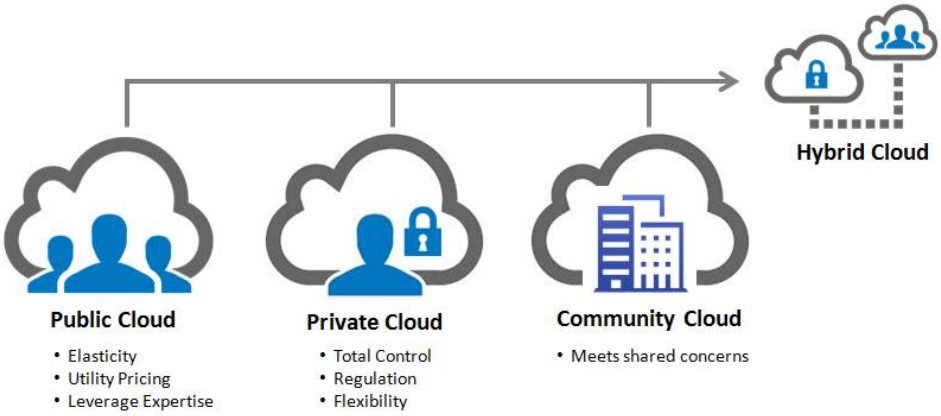
layanan tetap berkerja atau mampu pulih dalam waktu yang cepat saat terjadi gangguan.

2.1.2 Cloud Computing

Cloud computing merupakan metode komputasi yang memanfaatkan internet untuk menyediakan layanan teknologi informasi secara *virtual*. Melalui metode ini proses komputasi dialihkan ke server di jaringan internet sehingga komputer lokal tidak perlu lagi melakukan komputasi berat. *User* dapat mengakses layanan TI melalui antarmuka web yang disediakan oleh *web server* di internet untuk kebutuhan penyimpanan dan pemrosesan data. Cloud computing memiliki 3 model layanan yaitu, *Platform as a Service* (Paas), *Software as a Service* (SaaS), *Infrastructure as a Service* (IaaS). *Platform as a service* berarti *cloud* menyediakan lingkungan untuk pengujian, pengembangan, serta penyebaran aplikasi yang siap pakai berupa sistem operasi, *database, runtime* dan alat pengembangan sehingga pengembang tidak perlu mengelola infrastruktur secara lansung. *Software as a service* berarti cloud menyediakan *software* dan aplikasi sepenuhnya melalui internet, sehingga pengguna cukup mengakses melalui browser tanpa harus instalasi, *maintenance*, maupun *update* perangkat.

Infrastructure as a service berarti *cloud* menyediakan server, *storage*, dan jaringan dalam bentuk virtual sehingga pengguna dapat menyewa dan mengatur infrastruktur sesuai kebutuhan tanpa harus membeli ataupun mengelola perangkat keras secara lansung. Penelitian oleh Setiawan dkk., (2022) dengan judul, "Analisa Layanan Cloud Computing di Era Digital " menjelaskan bahwa dalam implementasinya *cloud computing* memiliki empat model seperti pada Gambar 2. *1* yang berasal dari browser https://sis.binus.ac.id/ yaitu *public cloud, private cloud, hybrid cloud*, dan *community cloud*. *Public cloud* berarti layanan *cloud* dapat diakses secara umum dan dapat digunakan secara bersama-sama sehingga pengguna tidak mengeluarkan biaya untuk investasi, *maintenance* maupun membangun infrastruktur aplikasi. *Private cloud* berarti layanan *cloud* yang digunakan secara eksklusif di satu organisasi dan aksesnya terbatas hanya untuk organisai tersebut. *Hybrid cloud*

merupakan gabungan *public cloud* dan *private cloud* terdapat pembagian hak akses data dan aplikasi yang digunakan, untuk beban kerja umum menggunakan layanan *public cloud*, sedangkan data sensitif menggunakan *private cloud.* Dan yang terakhir *community cloud* merupakan jenis layanan cloud yang digunakan bersama-sama oleh beberapa organisasi didalam satu komunitas yang memiliki kepentingan, tujuan, dan kebutuhan yang sama.



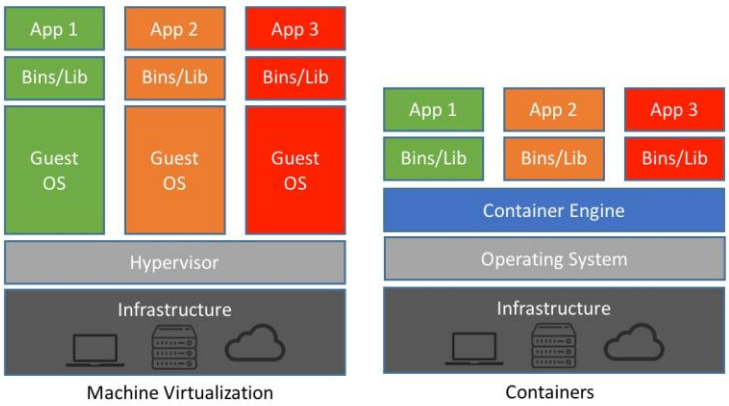
Gambar 2. 1 Model *Cloud Computing*[1]

2.1.3 Containerisasi

Containerisasi merupakan suatu metode virtualisasi pada level system operasi, dimana seluruh aplikasi beserta komponennya dikemas dalam satu unit yang disebut *container*. Container bekerja dengan berbagi kernel OS *host* dan hanya membawa aplikasi beserta komponen yang diperlukan, berbeda dengan *virtual machine* (VM) yang membutuhkan sistem operasi lengkap diatas *hypervisor* seperti pada Gambar 2. *2* yang bersumber dari https://www.netapp.com/blog/containers-vs-vms/. Hal ini menjadi keunggulan bagi *container* dikarenakan lebih ringan, cepat djalankan, dan efisien sumber daya sehingga metode ini banyak digunakan dalam pengembangan perangkat lunak.

Keunggulan utama *container* adalah mampu menyediakan lingkungan yang konsisten dari tahap pengembangan hingga produksi, serta mendukung orkestrasi skala besar dengan *tools* seperti kubernetes yang mampu mengelola ribuan *container*.

Penelitian oleh Santoso (2023), dengan judul "Tinjauan Pustaka *Kubernetes* Container Management" mengemukakan bahwa kubernetes *container* memiliki kemampuan mengelola aplikasi berbasis *container* secara efisien dengan beban komputasi yang cenderung rendah namun kinerja layanan tinggi.

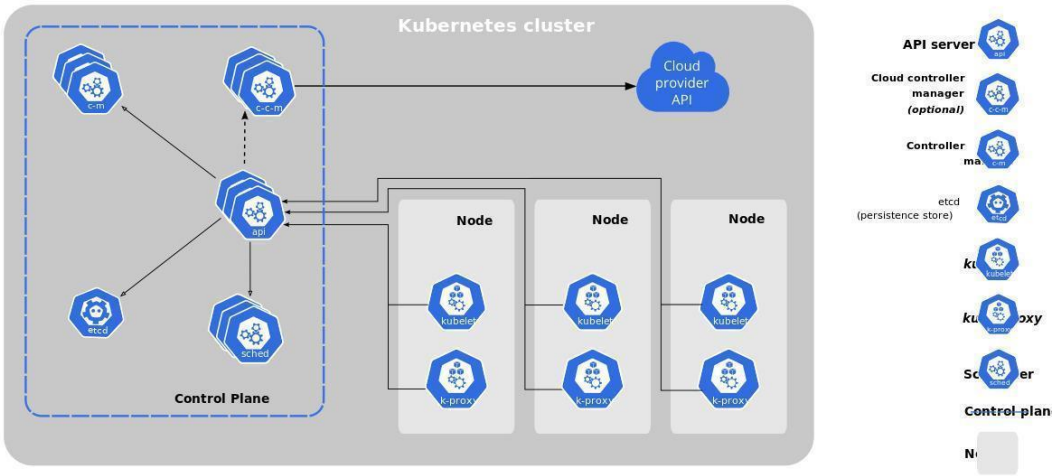


Gambar 2. 2 *Virtual Machine vs Container*[2]

2.1.4 Kubernetes

Kubernetes merupakan sistem orkestrasi kontainer bersifat *open source* yang dikembangkan oleh google dan kini dikelola oleh *Cloud Native Computing Foundation*. Platform ini digunakan untuk melakukan manajemen workloads aplikasi/layanan berbasis kontainer, dimana kubernetes menyediakan konfigurasi dan otomatisasi *service*/layanan secara deklaratif. kubernetes menyediakan beberapa fitur seperti *deployment*, scaling, load balancing, logging, dan monitoring, yang memungkinkan pengelolaan aplikasi secara efisien dan stabil sesuai kebutuhan sistem. Menurut Wiawan (2024) kubernetes memiliki keunggulan utama dalam menangani *load balancing* serta *auto scaling*, dimana teknologi ini akan mendistribusikan beban kerja ke beberapa *pod* secara otomatis, sehingga jika terjadi lonjakan trafik aplikasi tetap memiliki performa yang optimal. *Kubernetes* juga memiliki mekanisme *self-healing* yang bertujuan untuk mendeteksi serta menggantikan *pod* yang gagal secara otomatis agar layanan tetap tersedia.

Kubernetes memiliki dua komponen utama seperti pada Gambar 2. 3 yang bersumber dari https://www.geeksforgeeks.org/ yaitu *Control Plane* dan *Node. Control Plane* bertugas atas pengambilan keputusan pada *cluster*, seperti penjadwalan serta pengelolaan status *cluster*. Komponen *control plane* meliputi kube-apiserver yang bertindak sebagai antarmuka utama API, etcd yang bertugas sebagai penyimpanan data klaster, kube-scheduler yang bertugas dalam penjadwalan *pod*, kube-controller manager yang bertugas mengelola *node* dan job, dan *cloud-controller manager* yang bertugas mengintegrasikan sistem dengan penyedia *cloud*. Sementara itu, *node* juga memiliki beberapa komponen seperti kubelet yang berfungsi memastikan kontainer didalam *pod* berjalan sesuai spesifikasi, kube-proxy bertugas menangani jaringan antar-*pod* dan antar-layanan, serta Runtime kontainer seperti *container*d, yang bertugas menjalankan kontainer itu sendiri di dalam *node*. Selain komponen utama, terdapat juga AddOn seperti Dashboard dan DNS. AddOn ini biasanya dijalankan dalam *namespace* khusus bernama kube-system.

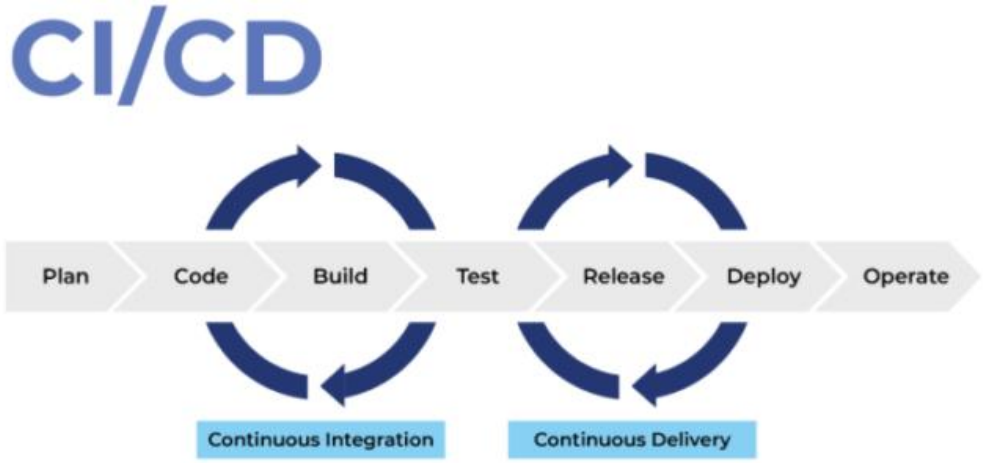


Gambar 2. 3 Komponen *kubernetes cluster*[3]

2.1.5 Deployment

deployment dari segi teknologi informasi merupakan proses menerapkan atau menjalankan aplikasi maupun sistem yang sudah dibangun ke lingkungan operasional

agar bisa digunakan oleh pengguna. Proses ini merupakan tahap akhir dari proses pengembangan sehingga aplikasi yang sudah selesai dikembangkan di aplikasikan ke lingkungan nyata. Dalam perkembangan teknologi saat ini *deployment* aplikasi banyak dilakukan menggunakan teknologi *container* yang dikombinasikan dengan system orkestrasi seperti kubernetes agar mendukung otomatisasi, skalabilitas, dan efisiensi operasional. Penelitian yang dilakukan oleh Gokarna (2021), dengan judul "DevOps Phases across Software Development Lifecycle" mengemukakan bahwa terdapat 4 proses yang mendukung keberhasilan *deployment* yaitu *build automation, release management, deployment automation, dan continous monitoring*. Selanjutnya *deployment* juga terbagi menjadi beberapa jenis, yaitu *manual deployment, automated deployment, rolling deployment, blue-green deployment, dan canary deployment. Manual deployment* dilakukan oleh admin/*developer*, seperti pada Gambar 2. *4* yang bersumber dari https://blogs.perficient.com/ automated *deployment* dilakukan oleh system (CI/CD pipeline), *rolling deployment* mengganti versi lama ke versi baru tanpa *downtime* secara bertahap, *blue-green deployment* berarti terdapat 2 environtment yang digunakan yaitu satu yang bertindak sebagai aktif dan yang lain siaga sebagai *backup*, dan *canary deployment* menjalankan aplikasi ke sebagian kecil pengguna untuk diuji performanya.

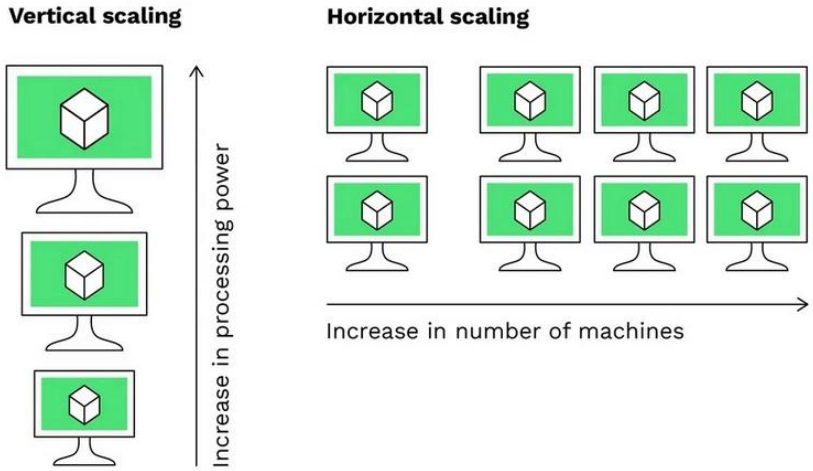


Gambar 2. 4 *Automated deployment* [4]

2.1.6 Auto Scaling

Auto scaling merupakan proses penyesuaian sumber daya komputasi (seperti server) berdasarkan permintaan beban kerja yang diterima. Tujuan utama dari auto scaling adalah memastikan aplikasi atau sistem dapat bekerja secara normal meskipun terdapat lonjakan permintaan dan sistem dapat mengoptimalkan sumber daya ketika permintaan rendah. Teknologi ini banyak digunakan pada lingkungan *cloud computing* dan *container orchestration* seperti *Kubernetes* untuk menjaga ketersediaan (availability) dan efisiensi operasional. Auto scaling bekerja dengan memantau beberapa metrik seperti penggunaan CPU, memori, jumlah permintaan per detik, atau antrian proses, kemudian menambah atau mengurangi *pod* secara otomatis sesuai aturan yang telah ditentukan.

Auto scaling terbagi menjadi dua jenis utama seperti pada Gambar 2. 5 yang bersumber dari https://dev.to/somadevtoo/, yaitu *horizontal scaling (scale out / scale in*) dan *vertical scaling (scale up/scale down*. Horizontal scaling berfungsi menambah atau mengurangi jumlah *pod*/instance ketika terjadi lonjakan permintaan. Vertical scaling berarti meningkatkan kapasitas sumber daya (CPU, RAM, penyimpanan) pada *pod* yang sudah ada. Selain dua jenis utama di atas terdapat juga beberapa jenis auto scaling yaitu *predictive autoscaling, scheduled autoscaling,* dan *reactive auto scaling. Predictive autocaling* menggunakan histori dan prediksi algoritma untuk mengantisipasi lonjakan permintaan serta menyesuaikan kapasitas sumber daya sebelum beban meningkat Xu dkk., (2025). *Scheduled autoscaling* menyesuaikan sumber saya pada jadwal yang telah ditentukan. *Reactive autoscaling* menyesuaikan kapasitas sumber daya berdasarkan metrik kinerja yang teramati contohnya penggunaan CPU atau memori.

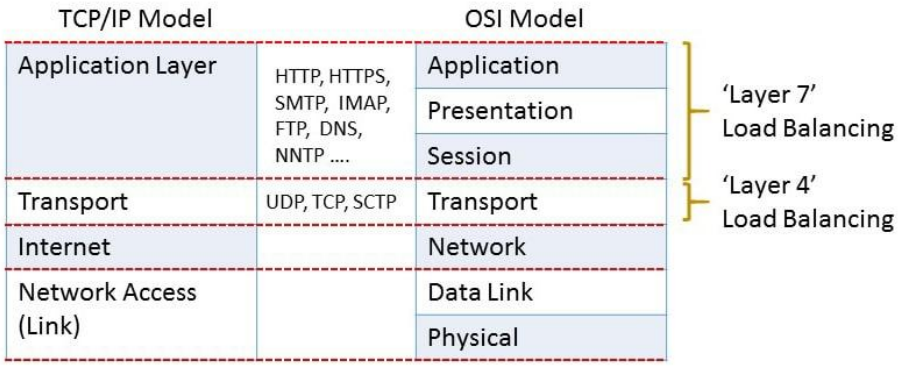


Gambar 2. 5 Vertical Scaling vs Horizontal Scaling[5]

2.1.7 Load Balancing

Load balancing merupakan teknik yang digunakan untuk mendistribusikan beban kerja secara merata ke beberapa sumber daya komputasi seperti server. Load balancing diperlukan untuk menjaga performa layanan/aplikasi, meningkatkan throughput, mengurangi latensi, serta menjamin ketersediaan layanan. Pada konteks cloud computing dan *container* orchestration *load balancing* berfungsi sebagai *traffic controller* yang memastikan permintaan user didistribusikan ke *node* yang paling optimal. Penelitian oleh Talukdar dkk., (2024) melakukan evaluasi efektivitas, performa, dan produktivitas teknik load balancing dengan berbagai algoritma, dan menyimpulkan bahwa load balancing tidak hanya berfungsi mendistribusikan beban kerja namun juga harus mampu beradaptasi terhadap konidisi sistem yang berubah-ubah. *Load balancing* dapat dilakukan di lapisan OSI layer 4 dan 7 seperti pada Gambar 2. 6 yang bersumber dari https://www.sjpas.com/. Pada layer 4 (trans*port* layer) menggunakan protocol TCP/UDP untuk membagi trafik berdasarkan informasi *port* dan Alamat ip. Pada layer 7 berbasis dns menggunakan *Domain Name System* untuk membagi beban kerja berdasarkan alamat ip yang diberikan klien. Pada layer 7

application layer memanfaatkan URL, cookie, atau header HTTP untuk menentukan distribusi trafik.



Gambar 2. 6 Load Balancing pada lapisan OSI[6]

2.1.8 Store Management

Dalam konteks *cloud computing, store management* merujuk pada praktik pengelolaan penyimpanan data yang mencakup aspek kemanan, efisiensi biaya, skalabiitas, dan ketersediaan. Salah satu pendekatan penting dalam melakukan store management adalah penerapan *lifecyle policies* untuk *automate data transitions,* dimana data akan dipindahkan secara otomatis ke penyimpanan yang lebih hemat biaya berdasarkan usia atau frekuensi aksesnya. Dalam *cloud store management* aspek keamanan menjadi fondasi penting. Seperti encryption untuk melindungi data dari akses yang tidak sah. Lalu implementasi *Identity and Access Management* (IAM) atau *Role-Based Access Control* (RBAC) guna memastikan hanya pengguna yang berwenang yang bisa mengakses data. Untuk memudahkan pengelolaan data di cloud dan memastikan berbagai layanan penyimpanan bisa berkomunikasi satu sama lain, terdapat standar bernama *Cloud Data Management Interface* (CDMI). Standar ini seperti "bahasa bersama" yang mengatur cara sistem penyimpanan cloud mengelola data, termasuk mengatur metadata (informasi tambahan tentang data), mengontrol siapa yang bisa mengakses, menentukan berapa lama data disimpan, serta mengatur

kualitas layanan (QoS). Dengan CDMI, layanan dari penyedia *cloud* yang berbeda bisa saling mendukung dan bekerja sama dengan lancar.

2.1.9 Otomasi

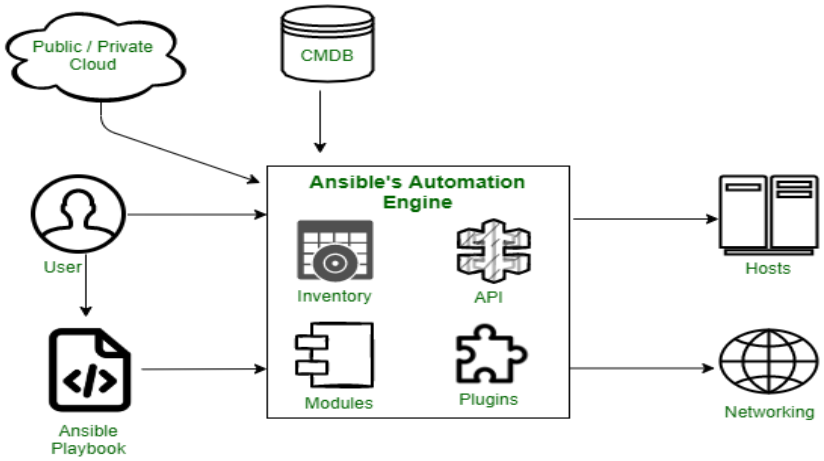
Otomasi merupakan proses penggunaan teknologi, perangkat lunak, atau mesin untuk menjalankan tugas tertentu secara mandiri. Dari segi teknologi, kata otomasi merujuk pada penggunaan teknologi atau *tools* untuk menjalankan tugas tanpa campur tangan manusia yang berkelanjutan. Otomasi dalam manajemen infrastruktur TI mencakup penggunaan *tools* untuk mengelola dan menyediakan konfigurasi, serta pemantauan sistem secara berkala dan otomatis. Tujuannya adalah untuk efisiensi operasional, mengurangi human error, dan mempercepat respon dalam menangani problem yang ada ketika mengelola sistem TI. Sitompul dkk., (2025) dalam penelitiannya menemukan bahwa teknologi otomasi memiliki dampak signifikan dalam meningkatkan produktivitas dan efisiensi operasional, seperti meminimalkan human error dan kemampuan optimalisasi waktu.

Dalam melakukan manajemen infrastruktur TI terdapat beberapa *tools* yang bisa digunakan yaitu *Ansible*, Teraform, Chef, dan Puppet. Pemanfaatan berbagai *tools* ini digunakan untuk manajemen konfigurasi, Infrastructure as a code, serta monitoring & incident response secara otomatis. Pemanfaatan otomasi juga bisa terintegrasi dengan CI/CD pipeline sehingga memungkinkan melakukan proses provisioning infrastruktur, *deployment*, aplikasi, serta secara otomatis memantau system ketika terjadi perubahan pada kode program. Penelitian oleh Putu Hariyadi dkk., (2023) menunjukkan bahwa integrasi *Ansible playbook* dengan gitlab CI/CD pipeline dapat digunakan untuk otomasi konfigurasi infrastruktur jaringan berbasis mikrotik dan berdasarkan uji coba yang dilakukan integrasi ini mampu mempercepat perubahan konfigurasi serta meminimalkan kesalahan konfigurasi. Hal ini membuktikan bahwa pemanfaatan otomasi dalam infrastruktur TI dapat meningkatkan kualitas layanan, menjaga konsistensu konfigurasi, serta skalabilitas system.

2.1.10 Ansible

Ansible merupakan sebuah tool open-source untuk otomatisasi manajemen infrastruktur TI yang dikembangkan oleh Red Hat. Alat ini dapat berperan sebagai otomatisasi konfigurasi sistem, *deployment* aplikasi, serta pengelolaan server termasuk pembaruan aplikasi/layanan. *Ansible* bekerja dengan prinsip infrastructure as Code (IaC), dimana infrastruktur didefinisikan melalui file konfigurasi kemudian dieksekusi untuk membangun dan mengelola sistem secara konsisten. Dalam penggunaannya tool ini menggunakan bahasa pemrograman berbasis YAML yang disebut *playbook* yang dirancang agar memudahkan manusia untuk membacanya. *Ansible* bekerja dengan cara menghubungkan serta mengeksekusi perintah pada server menggunakan SSH pada sistem operasi Linux.Unix atau WinRM (Windows Remote Management) pada sistem operasi Windows.

Berdasarkan pada Gambar 2. *7* yang bersumber dari https://btech.id/id/news/ proses kerja *Ansible* dimulai dari user yang membuat dan menjalankan *playbook*. Lalu *playbook* ini kemudian diproses oleh *Ansible* Automation Engine yaitu inti dari sistem *Ansible*. Di dalam mesin ini terdapat beberapa komponen penting, yaitu *inventory*, modules, plugins, API, CMBD, cloud, dan Networking. Inventory, berfungsi menyimpan daftar *host* dan kelompok *host* yang menjadi target otomatisasi. Modules, sebagai komponen inti yang mengeksekusi fungsi tertentu pada *host*, seperti instalasi paket atau pengaturan konfigurasi. Plugins, berfungsi untuk memperluas fungsi dan fitur tambahan. API, yang memungkinkan integrasi dengan sistem lain. CMDB (Configuration Management Database), yang menyediakan informasi konfigurasi dan status dari lingkungan TI. Cloud publik/privat, *ansible* dapat terintegrasi sebagai bagian dari infrastruktur yang dikelola. Dan networking, menunjukkan bahwa *Ansible* juga dapat digunakan untuk mengelola perangkat jaringan.



Gambar 2. 7 Proses *ansible*[7]

2.2 Penelitian Terdahulu

Dalam penelitian terdahulu ini diambil beberapa penelitian terdahulu yang sebelumnya telah dilakukan terkait dengan topik ini. Hal ini bertujuan untuk dapat melakukan perbandingan antara penelitian ini dengan penelitian sebelumnya, serta sebagai referensi untuk mendukung penyusunan penelitian ini. Beberapa penelitian terdahulu tersebut adalah sebagai berikut:

Penelitian yang dilakukan oleh Evianti dkk., (2021) dengan judul *Automation Provisioning* DevOps Website Server Menggunakan *Ansible* dan Vagrant. Penelitian ini menggunakan *Ansible* sebagai *tools* otomasi serta vagrant sebagai *virtual environment.* Penelitian ini berfokus pada *provisioning* server website untuk kebutuhan DevOps agar dapat berjalan secara otomatis dan konsisten. Hasil penelitian ini menunjukan bahwa *ansible* dapat melakukan penambahan server virtual secara otomatis dan efisien.

Penelitian yang dilakukan oleh Ramdhani dkk., (2023) dalam penelitiannya yang berjudul Automasi Konfigurasi Web Service pada Ubuntu Server Menggunakan *Ansible* Berbasis Python, Penelitian ini menggunakan *Ansible* sebagai *tools* otomasi serta ubuntu server sebagai sistem operasi. Penelitian ini berfokus pada penerapan *Ansible* untuk mengotomatisasi proses konfigurasi web server di Ubuntu. Hasil penelitian ini menunjukkan bahwa konfigurasi menggunakan *ansible* dapat dilakukan secara lebih cepat dan massive bila dibandingkan dengan konfigurasi menggunakan cara manual dan *service* dapat digunakan dengan baik.

Penelitian yang dilakukan oleh Kurniawan dkk., (2024)dengan judul Implementasi *Ansible* pada Otomasi Honeypot *deployment* Berbasis Web. Penelitian ini menggunakan *Ansible* sebagai *tools* otomasi serta Docker yang digunakan untuk menjalankan honeypot secara *containeri*zed. Penelitian ini berfokus pada penggunaan *Ansible* untuk mengotomatisasi proses instalasi dan *deployment honeypot* berbasis web. Dalam penelitian ini, hasilnya menunjukkan bahwa *ansible* dapat membuat proses *deployment honeypot* menampilkan logs sehingga bisa terpantau dengan baik.

Penelitian yang dilakukan oleh Hafidz (2024) yang berjudul Penerapan CI/CD dalam Pengembangan Web Aplikasi di *Kubernetes* dengan *Rancher*. Penelitian ini

mengeksplorasi penerapan *pipeline* CI/CD menggunakan GitLab pada lingkungan *Kubernetes* yang dikelola oleh Rancher. Fokusnya adalah bagaimana proses build, test, dan *deployment* aplikasi web dapat dilakukan secara otomatis. Hasil yang diperoleh adalah peningkatan efisiensi pengembangan serta penerapan DevOps yang lebih optimal melalui proses *deployment* yang berkelanjutan dan tanpa intervensi manual.

Penelitian yang dilakukan oleh Mubarak dkk., (2025) berjudul Implementasi Load Balancing pada *Cluster* Server Berbasis Raspberry Pi Memanfaatkan Algoritma Least Connection, Penelitian ini berfokus pada penerapan load balancing pada *cluster* server mini berbasis Raspberry Pi. Penelitian ini menggunakan *Ansible* dan Docker Swarm sebagai *tools* utama. Hasil penelitian menunjukkan bahwa beban kerja dapat didistribusikan secara merata antar *node* dengan memanfaatkan algoritma least connection, serta mempermudah pengelolaan klaster kecil secara terpusat.

Tabel 2. 1 Ringkasan penelitian terdahulu

Kategori Penulis	**Peneliti 1**	**Peneliti 2**	**Peneliti 3**	**Peneliti 4**	**Peneliti 5**
	Evianti dkk. (2021)	Jaya dkk. (2023)	Kurniawan, & Saputra (2024)	Hafidz (2024)	Budi dkk. (2025)
Kategori Tahun	2021	2023	2024	2024	2025
Judul	Automation Provisioning DevOps Website Server Menggunakan *Ansible* dan Vagrant	Automasi Konfigurasi Web Service pada Ubuntu Server Menggunakan *Ansible* Berbasis Python	Implementasi *Ansible* pada Otomasi Honeypot *deployment* Berbasis Web	Penerapan CI/CD dalam Pengembangan Web Aplikasi di *Kubernetes* dengan Rancher	Implementasi Load Balancing pada *Cluster* Server Berbasis Raspberry Pi Memanfaatkan Algoritma

					Least Connection
Kategori Penulis	**Peneliti 1**	**Peneliti 2**	**Peneliti 3**	**Peneliti 4**	**Peneliti 5**
Fokus Penelitian	Provisioning server website untuk DevOps	Automasi konfigurasi web server	Otomatisasi instalasi honeypot berbasis web	CI/CD pipeline di lingkungan *Kubernetes*	*Load balancing* server *cluster* mini
Teknologi/Tools	*Ansible*, Vagrant	*Ansible*, Ubuntu Server	*Ansible*, Docker	GitLab, Rancher K8s Engine	*Ansible*,Docker swarm
Hasil	Penyediaan infrastruktur DevOps menjadi lebih cepat dan replikatif	Proses konfigurasi lebih cepat dan konsisten	*deployment* honeypot menjadi otomatis dan efisien	Proses *deployment* menjadi otomatis dan berkelanjutan	Pendistribusian beban antar *node* menjadi merata

Berdasarkan kajian terhadap beberapa penelitian terdahulu pada , diketahui bahwa berbagai penelitian telah membahas penerapan *Ansible* sebagai *tools* otomasi pada berbagai bidang, seperti provisioning server, konfigurasi web *service*, *deployment* honeypot, implementasi pipeline CI/CD pada *Kubernetes*, serta pengelolaan *cluster* server berbasis Docker Swarm. Hasil dari penelitian-penelitian tersebut menunjukkan bahwa *Ansible* mampu meningkatkan efisiensi, mempercepat proses konfigurasi dan *deployment*, mengurangi kesalahan akibat konfigurasi manual, serta mempermudah pengelolaan infrastruktur secara terpusat. Meskipun demikian, sebagian besar penelitian tersebut masih berfokus pada aspek otomasi tertentu, seperti provisioning server, konfigurasi layanan, *deployment* aplikasi, maupun implementasi CI/CD, dan belum membahas otomasi pengelolaan *Kubernetes Cluster* secara menyeluruh yang mencakup proses *deployment*, *autoscaling* , load balancing, serta manajemen *storage* dalam satu mekanisme otomasi.

Dibandingkan penelitian-penelitian terdahulu, penelitian ini memiliki fokus spesifik pada penyederhanaan manajemen infrastruktur TI melalui otomasi konfigurasi

Kubernetes cluster menggunakan *Ansible*. Dengan pendekatan ini, proses *deployment* dan pengelolaan *node* dalam *cluster Kubernetes* dapat dilakukan secara lebih cepat, konsisten, dan terpusat, yang menjadi solusi terhadap kompleksitas manajemen infrastruktur. Penelitian ini berfokus pada penerapan *Ansible* sebagai *tools* otomasi untuk mengelola *Kubernetes Cluster* secara terintegrasi. Otomasi yang dilakukan tidak hanya mencakup proses instalasi dan *deployment cluster*, tetapi juga pengelolaan aplikasi melalui *deployment*, implementasi *autoscaling* menggunakan Horizontal *pod* Autoscaler (HPA), load balancing, serta manajemen *storage* menggunakan nfs-server. Dengan mengintegrasikan seluruh proses tersebut ke dalam *playbook Ansible*, pengelolaan infrastruktur menjadi lebih konsisten, dan memudahkan dalam hal pemeliharaan.