BAB 2
TINJAUAN PUSTAKA

2.1 Landasan Teori

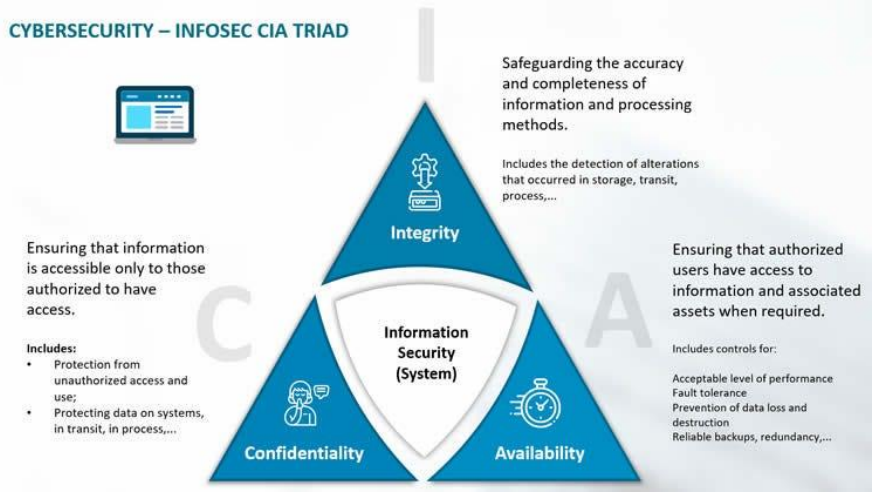
2.1.1 Otomatisasi

Otomatisasi adalah proses penggunaan sistem atau perangkat lunak untuk melaksanakan tugas berulang secara otomatis tanpa campur tangan manusia secara langsung. Dalam konteks teknologi informasi, otomatisasi pada Gambar 2.1 (sumber: Aspriyono, 2018) bertujuan untuk meningkatkan efisiensi operasional, mengurangi kesalahan manusia (*human error*), serta memastikan konsistensi dalam pengelolaan infrastruktur TI (Drosos et al., 2024).

Gambar 2.1 Otomatisasi

2.1.2 Keamanan

Keamanan sistem informasi adalah praktik yang bertujuan untuk melindungi sistem dan data dari akses, penggunaan, pengungkapan, gangguan, modifikasi, atau perusakan yang tidak sah. Dalam dunia digital, keamanan menjadi aspek fundamental karena meningkatnya serangan siber dan kerentanan sistem, terutama pada infrastruktur TI yang terhubung ke internet. Elemen utama dalam keamanan informasi dapat dilihat pada Gambar 2.2 (sumber: Make Computer Science Great Again, 2023) yaitu mencakup kerahasiaan (confidentiality), integritas (integrity), dan ketersediaan (availability) atau dikenal dengan istilah CIA triad. Menurut Wibowo et al. (2024), salah satu strategi penting untuk menjaga keamanan sistem adalah melakukan pembaruan patch keamanan secara rutin guna menutup celah (vulnerability) yang dapat dimanfaatkan oleh pihak yang tidak bertanggung jawab.



Gambar 2.2 Ilustrasi Konsep *CIA Triad* dalam Keamanan Jaringan

2.1.3 Ansible

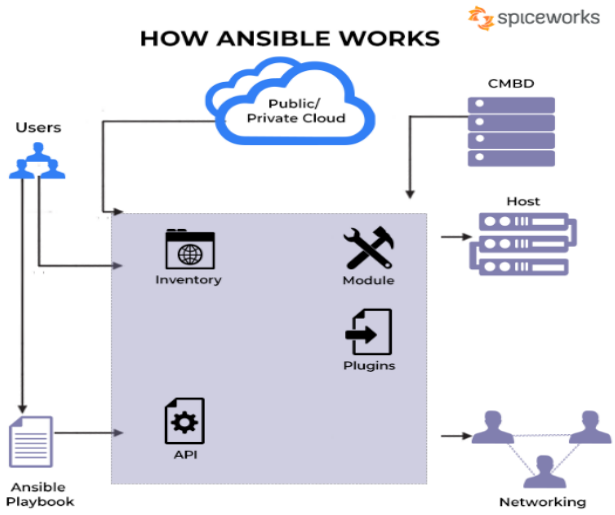
Ansible merupakan perangkat lunak open-source yang digunakan untuk otomatisasi infrastruktur Teknologi Informasi (TI), mencakup penyediaan infrastruktur (*infrastructure provisioning*), pengelolaan konfigurasi (configuration management), orkestrasi, serta otomatisasi aplikasi dan keamanan (*application deployment & security automation*). Menurut Heap (2016), Ansible mengusung arsitektur *agentless*, yaitu tidak memerlukan penginstalan perangkat lunak *agent* atau daemon khusus pada server target (*managed nodes*). Seluruh instruksi dan komunikasi antar-sistem dikelola secara terpusat dari *Control Node* dengan memanfaatkan protokol standar SSH (pada sistem Unix/Linux) atau WinRM (pada sistem *Windows*). Berdasarkan panduan teknis Heap (2016), Ansible menggunakan bahasa spesifikasi deklaratif berbasis YAML (YAML *Ain't Markup Language*) yang dikemas dalam bentuk *playbook*. *Sintaks* YAML dirancang agar mudah dibaca oleh manusia (*human-readable*) sehingga mempermudah administrator dalam mendefinisikan *state* atau kondisi ideal sistem. Selain itu, Ansible menerapkan sifat idempoten (*idempotency*), di mana eksekusi playbook secara berulang pada server target hanya akan menerapkan perubahan jika terdapat ketidaksesuaian state, sehingga menjamin konsistensi, presisi, dan efisiensi pengelolaan keamanan jaringan maupun *patching* sistem secara berkelanjutan.

Ansible memiliki beberapa komponen di antaranya :

1) *Inventory*, berisi alamat *IP* dari setiap perangkat yang dikelola.

2) *Modules*, sebuah package yang berisi unit kode untuk dieksekusi serta memiliki fungsi yang spesifik.
3) *Ad-hoc*, baris perintah untuk menjalankan *task* melalui *Command Line Interface* atau CLI.
4) *Playbooks*, sebuah file dengan format *YAML* yang berisi serangkaian *tasks* yang dieksekusi secara urut.

Selain itu, terdapat komponen lain seperti *roles* yaitu suatu metode untuk memecah kompleksitas *playbook* menjadi lebih kecil yang direpresentasikan ke dalam bentuk komponen modular yang di dalamnya memuat gabungan dari beberapa konten dengan fungsi spesifik Kurniawan & Saputra (2024).



Gambar 2.3 *Ansible*

Dapat dilihat pada gambar 2.3 (sumber: spiceworks.com) salah satu keunggulan utama Ansible dibandingkan alat otomasi lainnya seperti *Puppet*, *Chef*, atau *SaltStack* adalah sifatnya yang *agentless*, artinya Ansible tidak memerlukan *agen* atau *service* tambahan yang berjalan di sistem target. Ini membuatnya lebih ringan, aman, dan lebih mudah diintegrasikan dalam lingkungan yang heterogen, baik itu infrastruktur *on-premise* maupun *cloud* seperti *GCP* (*Google Cloud Platform*). Penggunaan SSH sebagai kanal komunikasi utama juga menyederhanakan manajemen konektivitas karena tidak memerlukan *port* tambahan atau instalasi *daemon* khusus di sisi klien. Dalam skenario *hybrid cloud*, Ansible dapat dikombinasikan dengan *dynamic inventory* untuk mendeteksi *host virtual* dari penyedia *cloud* secara otomatis, memungkinkan orkestrasi lintas

infrastruktur secara efisien (Muppala, 2025).

Selain itu, *YAML-based syntax* dalam *playbook* menjadikan Ansible sangat mudah dipahami, bahkan oleh administrator sistem yang tidak memiliki latar belakang pemrograman mendalam. Fitur *idempotensi* Ansible, yaitu kemampuan untuk menjalankan tugas berkali-kali dengan hasil yang konsisten, menjamin keamanan proses otomasi konfigurasi dan pembaruan. Dalam praktik *DevOps* dan *SecOps modern*, Ansible tidak hanya digunakan untuk konfigurasi awal *server*, tetapi juga untuk *continuous compliance*, *security hardening*, dan *patch automation*. Dengan ekosistem modul yang luas dan komunitas aktif, Ansible menjadi alat yang ideal untuk pengelolaan sistem skala besar secara terstandarisasi dan terotomatisasi.

2.1.4 Patch Management

Patch Management adalah proses pengelolaan pembaruan perangkat lunak (*patch*) untuk memperbaiki kerentanan keamanan, meningkatkan performa, dan memperbaiki bug dalam sistem. Proses ini meliputi identifikasi, penilaian, pengujian, penerapan, serta dokumentasi *patch* yang relevan guna memastikan sistem tetap aman dan stabil (Aryono & Hilal, 2018). Fokus utama dari *patch* management adalah *security patch*, yaitu *patch* yang dirilis untuk menutup kerentanan keamanan yang berpotensi dieksploitasi oleh pihak tidak bertanggung jawab.

Menurut Bellamkonda (2023), security patch merupakan komponen vital dalam strategi keamanan jaringan, karena mampu mencegah insiden siber yang berasal dari eksploitasi celah yang telah diketahui. *Patch* jenis ini biasanya bersifat kritis dan perlu diterapkan segera setelah dirilis oleh *vendor*, guna menghindari risiko kompromi sistem. Islavath (2020) menambahkan bahwa dalam praktik modern berbasis *Infrastructure as Code* (IaC), pengelolaan *security patch* tidak lagi dilakukan secara manual, melainkan diotomatisasi dalam *pipeline DevOps*. Pendekatan ini mempercepat respons terhadap ancaman keamanan dan menjaga konsistensi konfigurasi sistem.

Dalam konteks otomasi, penggunaan Ansible telah terbukti meningkatkan efisiensi dan keandalan dalam pengelolaan *patch*. Ansible memungkinkan

administrator untuk menyusun skrip otomatis (*playbook*) yang dapat menjalankan proses pembaruan pada banyak sistem secara bersamaan, memantau status patch, dan melakukan verifikasi pasca-instalasi secara terstruktur. Hal ini tidak hanya menghemat waktu, tetapi juga mengurangi kesalahan manual dan memastikan bahwa seluruh sistem mengikuti kebijakan keamanan yang sama.

Tahapan *Patch Management* pada Gambar 2.4 (sumber: techpio.com) terdiri dari:

1) Inventarisasi Sistem dan Aplikasi

   Mengidentifikasi semua perangkat lunak dan infrastruktur yang memerlukan patch, termasuk sistem operasi, aplikasi, dan dependensinya. Mencantumkan semua *host* kedalam *inventory*, ini memastikan semua *node* yang akan dipatch sudah teridentifikasi dan siap untuk otomatisasi.

2) Pemantauan Rilis *Patch*

   Memantau informasi rilis patch dari vendor untuk mendeteksi pembaruan baru dengan *apt list –upgradable*.

3) Evaluasi dan Prioritas

   Menilai urgensi (prioritas) setiap *patch* berdasarkan tingkat keparahan kerentanan dan eksposur sistem.

4) Pengujian *Patch* (*Testing*)

   Menerapkan patch di lingkungan uji (*staging environment/snapshot*) untuk memastikan tidak menyebabkan gangguan terhadap sistem produksi (stabilitas dan performa).

5) Penerapan Patch (*Deployment*)

   Melakukan *deployment* (proses penerapan) patch ke sistem produksi secara otomatis menggunakan alat Ansible.

6) Verifikasi dan Dokumentasi

   Melakukan validasi bahwa *patch* berhasil diterapkan dan mendokumentasikan aktivitas sebagai bagian dari audit keamanan dan kepatuhan.

Gambar 2.4 *Patch Management Process*

2.1.5 Ubuntu

Ubuntu adalah distribusi *Linux* berbasis *Debian* yang bersifat *open-source* dan gratis, pertama kali dirilis pada tahun 2004. *Ubuntu* hadir dalam tiga varian: *Desktop*, *Server*, dan *Core*, yang semuanya dapat dijalankan di komputer *standalone* maupun *mesin virtual*. Khusus untuk *Ubuntu Server*, distribusi ini banyak dipakai di lingkungan *server* berkat kestabilannya, keamanannya, dan dukungan kuat dari komunitas (Patoni et al., 2024)

2.1.6 Google Cloud Platform

Google Cloud Platform atau biasa disingkat GCP merupakan salah satu penyedia terkemuka layanan *cloud computing* yang ditawarkan oleh *Google*. *Cloud computing*, penyimpanan, *cloud database*, analitik, pembelajaran mesin, kecerdasan buatan, *Internet of Things* (*IoT*), dan lainnya adalah beberapa layanan infrastruktur *cloud* yang disediakan oleh *Google Cloud Platform* (Kurniawan & Saputra, 2024).

Beberapa layanan yang tersedia di GCP pada Gambar 2.5 (sumber: k21academy.com) di antaranya sebagai berikut:

1) *Google Compute Engine*

   Google Compute Engine adalah layanan *cloud computing* yang memungkinkan *user* membuat dan mengelola mesin *virtual* di infrastruktur *Google*, yang memungkinkan user menjalankan berbagai jenis tugas. Salah satu layanannya

adalah *Virtual Machine Instances* yang nantinya akan digunakan untuk membuat *VM*.

2) *Google Cloud Storage*

   Google Cloud Storage adalah layanan penyimpanan *cloud* yang memungkinkan user menyimpan dan mengelola data secara skalabilitas dengan berbagai jenis penyimpanan, seperti penyimpanan objek dan penyimpanan berbasis blok, dan lainnya.

3) *Google BigQuery*

   Google BigQuery adalah layanan analitik data yang memungkinkan analisis data secara cepat dan skalabel.

Gambar 2.5 Layanan *Google Cloud Platform*

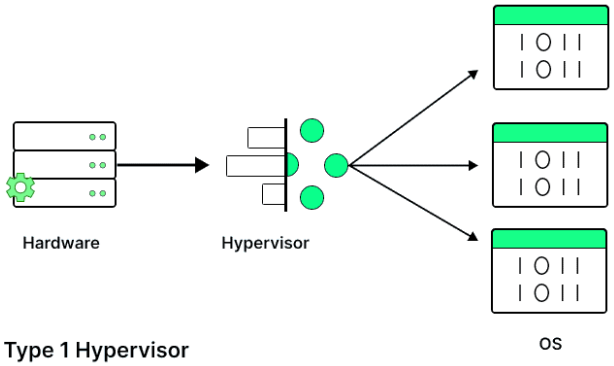
2.1.7 YAML

Serialisasi data yang sering kali berada di antara bahasa komputer yang paling populer. *YAML* biasanya digunakan sebagai format untuk file konfigurasi, tetapi kemampuan serialisasi objeknya menjadikannya alternatif potensial untuk bahasa seperti *JSON* atau *Python*. *YAML* digunakan sebagai bahasa pemrograman untuk membuat perintah dalam *Ansible-playbook* (Rafsyandjani, 2025).

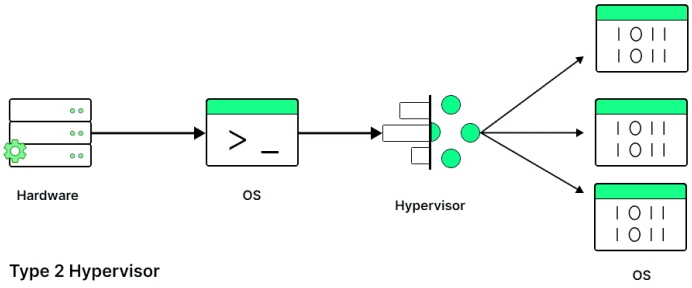
2.1.8 Virtual Machine

Virtual Machine (VM) merupakan sebuah sistem operasi atau aplikasi yang di *install* pada *hypervisor* dan memiliki fungsi layaknya perangkat fisik (*hardware*)

atau dapat juga disebut sebagai duplikat dari komputer asli. Virtualisasi adalah sebuah proses pembuatan sesuatu yang awalnya berbentuk fisik menjadi berbentuk *software* atau *virtual*. *Software* yang digunakan untuk virtualisasi dinamakan *hypervisor*. *Software* tersebut digunakan untuk membuat dan mengatur *Virtual Machine*. Selain itu, *hypervisor* juga dapat berperan sebagai agen penghubung antara *VM* dengan perangkat fisik. *Hypervisor* dibagi menjadi dua jenis yaitu *hypervisor* tipe 1, tipe ini berjalan langsung diatas perangkat keras dan *hypervisor* contoh untuk tipe 1 adalah *vsphere* dan *citrix xen server*. Tipe 2, tipe ini berjalan diatas *host* os contoh dari tipe ini adalah *oracle virtualbox* dan *vmware workstation*. Hal ini dapat dilihat pada Gambar 2.6 (sumber: lambdatest.com) *Type* 1 dan Gambar 2.7 *Type* 2. *Hypervisor* dapat menjalankan perangkat lunak apa pun yang berjalan pada perangkat keras *bare metal* sementara menyediakan isolasi dari perangkat keras yang sebenarnya (Firmansyah et al., 2019).



Gambar 2.6 *Type 1 Hypervisor*



Gambar 2.7 *Type 2 Hypervisor*

2.1.9 Inventory

Inventory dalam konteks keamanan siber adalah proses pendataan dan pengelolaan seluruh aset teknologi informasi, termasuk perangkat keras, perangkat lunak, aplikasi, dan jaringan yang dimiliki organisasi. Tujuannya adalah memastikan semua aset yang digunakan tercatat dengan baik untuk memudahkan pengawasan, pengelolaan risiko, dan penerapan kontrol keamanan (Laksana & Mulyani, 2024). Dengan inventarisasi yang baik, organisasi dapat dengan cepat mengidentifikasi aset rentan dan memastikan pembaruan serta perlindungan diterapkan secara tepat (Wibowo et al., 2024).

2.1.10 Playbook

Playbook dalam keamanan siber adalah dokumen panduan berisi prosedur standar (SOP) yang mendetail untuk menangani insiden keamanan siber. *Playbook* membantu tim keamanan bertindak cepat, konsisten, dan terkoordinasi saat menghadapi ancaman seperti serangan *malware*, *phishing*, atau pelanggaran data (Satriyawan & Susanto, 2023). Dengan adanya *playbook*, proses deteksi, respons, dan pemulihan dapat dilakukan lebih efektif dan efisien, serta mengurangi potensi kesalahan manusia (Laksana & Mulyani, 2024).

2.2 Penelitian Terdahulu

Dalam penelitian terdahulu ini diambil beberapa penelitian terdahulu yang sebelumnya telah dilakukan pengujian dan terdapat sembilan landasan teori yang digunakan.

Pada penelitian yang dilakukan oleh Afifi Al-Atsari & Suharjo (2023), dibahas integrasi *server on-premise* dengan *server cloud* menggunakan VPN IPsec untuk mengamankan komunikasi antara kedua *server*. Penelitian ini menunjukkan bahwa pengamanan komunikasi data antara *server cloud* dan lokal dapat dilakukan dengan menggunakan teknologi VPN yang berbasis IPsec, yang menggunakan *Mikrotik RouterOS* dan *Google Cloud Platform* (GCP). Pengujian yang dilakukan melibatkan simulasi dua *virtual machine* untuk menguji konektivitas, firewall, sniffing, dan ketahanan koneksi. Hasil dari penelitian ini menunjukkan bahwa penggunaan VPN IPsec memberikan solusi efektif dalam mengamankan

komunikasi antara *server on-premise* dan *cloud*, namun tidak menyentuh aspek otomatisasi sistem keamanan atau pembaruan *patch*.

Pada penelitian yang dilakukan oleh Adil & Beeh (2024), dikembangkan monitoring otomatis untuk sistem yang berbasis *on-premise* dan *cloud* dengan menggunakan teknologi *Jenkins*. Fokus utama dari penelitian ini adalah pemantauan sistem secara otomatis yang dilengkapi dengan pemberitahuan *email* ketika terjadi *error*. Pengujian dilakukan dengan mengintegrasikan monitoring *cloud* dan lokal, menggunakan *blackbox testing* untuk mengevaluasi efektivitas sistem. Hasil dari penelitian ini menunjukkan bahwa *Jenkins* dapat mengotomatisasi pemantauan serta memberikan notifikasi yang membantu administrator dalam melakukan tindakan cepat, namun penelitian ini lebih berfokus pada monitoring dan orkestrasi, bukan pada pembaruan atau manajemen keamanan sistem.

Pada penelitian yang dilakukan oleh Kurniawan & Saputra (2024), dieksplorasi penggunaan Ansible untuk otomatisasi *deployment honeypot* berbasis web untuk mendeteksi ancaman di *cloud* GCP. Dalam penelitian ini, Ansible, Flask, Docker, PostgreSQL, dan ReactJS digunakan untuk membangun dan mengelola *honeypot* di berbagai *region* GCP. Pengujian yang dilakukan melibatkan *deployment honeypot* otomatis, serta pengujian terhadap *SSH passwordless* dan *log* trafik serangan. Hasil penelitian ini menunjukkan bahwa *honeypot* otomatis dapat digunakan untuk mendeteksi ancaman dengan lebih cepat, namun penelitian ini lebih berfokus pada deteksi ancaman daripada otomatisasi *update patch* atau manajemen keamanan sistem.

Pada penelitian yang dilakukan oleh Craiu & Catrina (2024), dikembangkan skenario pengujian dan *prototyping* otomatis untuk solusi keamanan jaringan menggunakan virtualisasi dan Ansible. Penelitian ini bertujuan untuk membangun topologi jaringan virtual secara otomatis untuk menguji integritas dan *repeatability* konfigurasi keamanan. Pengujian dilakukan dengan menggunakan Ansible, Vagrant, VirtualBox, dan *Playbooks* untuk membangun dan mengonfigurasi jaringan *virtual*, serta menguji keamanan. Hasil dari penelitian ini menunjukkan bahwa penggunaan Ansible dalam skenario virtual dapat meningkatkan efektivitas

uji coba awal solusi keamanan, namun tidak berfokus pada penerapan *patch* atau *update* keamanan di sistem nyata.

Pada penelitian yang dilakukan oleh Chewe (2023), dirancang arsitektur keamanan otomatis untuk infrastruktur TI *hybrid cloud* dengan pendekatan *Zero Trust* dan *Security as Code*. Menggunakan *Infrastructure as Code* (IaC) dengan *Terraform*, IAM, SOAR, Kubernetes, dan tools DevOps lainnya, penelitian ini fokus pada desain arsitektur yang aman untuk sistem *hybrid*. Penelitian ini lebih berfokus pada strategi desain keamanan yang mengintegrasikan praktik dari berbagai penyedia *cloud* (*Microsoft*, AWS, GCP) daripada pada implementasi otomatisasi *update patch* keamanan menggunakan alat seperti Ansible.

Pada penelitian yang dilakukan oleh Gustian et al. (2023), dibahas implementasi *automation deployment* pada *Google Cloud Compute* VM menggunakan *Terraform*. Penelitian ini berfokus pada bagaimana *Terraform* dapat digunakan untuk mengotomatisasi proses *deployment virtual machine* di *Google Cloud Platform*. Penggunaan *Terraform* memungkinkan *deployment* yang konsisten, cepat, dan dapat diulang untuk sumber daya *cloud*. Implementasi melibatkan konfigurasi dan eksekusi *skrip Terraform* untuk mengelola siklus hidup *VM*. Penelitian ini sangat relevan dengan otomatisasi *deployment* infrastruktur *cloud*, namun fokus utamanya adalah pada *deployment VM* itu sendiri dan mungkin tidak membahas secara mendalam aspek keamanan atau *patch management* secara otomatis.

Pada penelitian yang dilakukan saat ini, merupakan pengembangan dari studi-studi sebelumnya dengan fokus pada implementasi *Ansible* untuk otomatisasi pembaruan patch keamanan pada sistem operasi *Linux* di lingkungan *on-premise* dan *cloud*. Penelitian ini menggunakan GCP VM *Instances* dan *Playbook* YAML untuk menerapkan serta mengonfigurasi parameter keamanan secara terpusat. Pengujian dilakukan dengan memverifikasi status *patch* serta menguji integritas dan konfigurasi sistem setelah pembaruan. Evaluasi dilakukan dengan mencatat keberhasilan *patch* yang terinstal, waktu proses otomatisasi, serta ketahanan sistem terhadap skenario simulasi serangan pasca *patching*. Penelitian ini memberikan pendekatan yang praktis dan dapat diterapkan dalam lingkungan TI nyata yang *hybrid*.

Tabel 2.1 Penelitian Terdahulu

Peneliti & Tahun	**Judul Penelitian**	**Pembahasan Utama**	**Metode / Tools**	**Cara Pengujian / Implementasi**	**Gap Penelitian**
HA Al-Atsari & I. Suharjo (2023)	Integrasi *Server On-Premise* dengan *Server Cloud* Menggunakan *Cloud VPN* dan *Mikrotik IPsec*	Pengamanan komunikasi jaringan antara *server cloud* dan lokal menggunakan *VPN IPsec*.	*Mikrotik RouterOS, VPN IKEv2, Google Cloud Platform*	Simulasi dua *VM (thanos-onprem & thanos-cloud)*, pengujian konektivitas, *firewall*, *sniffing*, dan ketahanan koneksi.	Fokus pada keamanan jalur VPN jaringan, belum menyentuh otomatisasi *patching* OS server.
SB Adil & YR Beeh (2024)	Implementasi Monitoring Sistem Perusahaan *On-Premises* dan *Cloud* Menggunakan Teknologi *Jenkins*	Monitoring otomatis sistem *hybrid* menggunakan *Jenkins* dan pengujian otomatis berbasis *email* notifikasi.	*Jenkins, GCP, PostgreSQL, Email Extension Plugin*	Simulasi integrasi monitoring *cloud* dan lokal dengan notifikasi *error* otomatis menggunakan *blackbox testing*.	Fokus pada pemantauan (*monitoring*), bukan pada otomatisasi *patching* dan *hardening* OS.
D. Kurniawan & Y.M. Saputra (2024)	Implementasi *Ansible* pada Otomasi *Honeypot Deployment* Berbasis *Web*	Otomatisasi *deployment honeypot* ke *cloud* GCP dengan *dashboard web* untuk memantau instalasi dan serangan.	*Ansible, Ansible AWX, Flask, Docker, PostgreSQL*, GCP, *ReactJS*	*Deployment honeypot* otomatis ke 3 *region cloud* GCP, *testing SSH passwordless, log traffic* serangan.	Terbatas pada *deployment* aplikasi *honeypot* di cloud, bukan *patching* OS terpusat di lingkungan *hybrid*.
G. Craiu & O. Catrina (2024)	*Fast Prototyping and Testing of Network Security Solutions Using Virtualization and Ansible*	Otomatisasi skenario keamanan jaringan *virtual* sebelum diterapkan di sistem nyata.	*Ansible, Vagrant, VirtualBox, Playbooks*	Pembangunan topologi *virtual* otomatis dan uji *repeatability* serta integritas konfigurasi keamanan jaringan.	Terbatas pada simulasi virtual lokal (Vagrant), belum diimplementasikan pada infrastruktur *cloud/hybrid* nyata.

Peneliti & Tahun	Judul Penelitian	Pembahasan Utama	Metode / Tools	Cara Pengujian / Implementasi	Gap Penelitian
Mutale Chewe (2021)	*Hybrid Cloud Infrastructure Security: Security Automation Approaches for Hybrid IT*	Rancangan arsitektur keamanan otomatis *hybrid cloud* dengan pendekatan *Zero Trust* dan *Security as Code*.	*IaC (Terraform), IAM, SOAR, Kubernetes, DevOps Tools*	Studi kasus *enterprise*: evaluasi praktik *Microsoft, AWS, GCP* dalam integrasi *cloud*-lokal dan orkestrasi keamanan.	Bersifat kerangka teoritis/arsitektural, tanpa implementasi teknis eksekusi *playbook* secara praktis.
Debi Gustian, Yuli Fitrisia, Sugeng Purwantoro E.S.G.S, Wenda Novayani (2023)	Implementasi *Automation Deployment* pada *Google Cloud Compute VM* menggunakan *Terraform*	Penelitian ini berfokus pada implementasi otomatisasi *deployment* pada *Google Cloud Compute VM* dengan menggunakan *Terraform*.	*Terraform, Google Cloud Compute VM*	Implementasi otomatisasi *deployment* dilakukan pada *Google Cloud Compute VM* menggunakan *Terraform*, diikuti dengan pengujian proses *deployment*.	Fokus pada *provisioning* infrastruktur (Terraform), bukan *configuration management* dan *patching* OS (Ansible).
Erdita Katri	Otomasi Pengelolaan dan Pembaruan Sistem Keamanan dengan *Ansible* di Infrastruktur TI Berbasis *On-Premise* dan *Cloud*	Penelitian ini fokus pada implementasi *Ansible* untuk otomatisasi pembaruan *patch* keamanan pada *OS Linux* di dua lingkungan (*on-premise & cloud*).	*Ansible, GCP VM Instance, Ubuntu, Playbook YAML*	Pengujian dilakukan dengan memverifikasi status *patch* menggunakan perintah sistem, serta menguji integritas dan konfigurasi sistem setelah pembaruan.	[Kebaharuan] Menerapkan otomatisasi *patching* OS & *hardening* keamanan terpusat berbasis Ansible di infrastruktur *hybrid* nyata (On-Premise & GCP).