BAB 2
TINJAUAN PUSTAKA

Bab ini membahas landasan teori dan kajian pustaka yang berkaitan dengan penerapan *dashboard self-service* untuk otomatisasi *provisioning virtual machine*. Landasan teori digunakan untuk menjelaskan konsep dan prinsip dasar yang menjadi pijakan penelitian, sedangkan kajian pustaka menyajikan penelitian-penelitian terdahulu yang relevan sebagai dasar perbandingan serta untuk mengidentifikasi celah penelitian yang diangkat dalam tesis ini.

2.1 Penelitian Terdahulu

Penelitian yang dilakukan Luolajan-Mikkola (2024) membahas penerapan *Infrastructure as Code* (IaC) pada lingkungan di luar *cloud* publik, dengan fokus pada penyediaan dukungan IaC pada *compute environment*, khususnya lingkungan *on-premises* yang tidak memiliki mekanisme IaC secara bawaan. Penelitian ini bertujuan mengidentifikasi kebutuhan utama sistem IaC melalui pendekatan *human-centric*, yang dilakukan melalui survei dan wawancara terhadap pengembang, serta analisis karakteristik sistem IaC yang telah ada. Berdasarkan kebutuhan yang teridentifikasi, penelitian ini merancang dan mengimplementasikan sebuah prototipe sistem IaC bernama Ink yang bersifat modular dan memungkinkan integrasi dengan berbagai platform komputasi melalui antarmuka API. Evaluasi dilakukan dengan menilai kesesuaian prototipe Ink terhadap kebutuhan pengguna serta melalui pengujian pengguna dalam lingkungan terkontrol. Hasil penelitian menunjukkan bahwa penerapan dukungan IaC pada lingkungan non-*cloud* dapat diimplementasikan dan mampu memenuhi kebutuhan pengembang dan organisasi, namun memerlukan upaya pengembangan perangkat lunak tambahan yang signifikan, sehingga kelayakannya bergantung pada skala dan prioritas organisasi yang mengadopsinya. Namun demikian, interaksi pada prototipe sistem Ink tersebut masih sepenuhnya bergantung pada antarmuka baris perintah (CLI) atau modifikasi kode. Ketiadaan lapisan antarmuka visual (*dashboard*) menjadikan solusi ini belum mampu mendukung konsep layanan mandiri (*self-service*) bagi pengguna awam, serta belum dilengkapi dengan fitur

keamanan preventif seperti otomatisasi kredensial akses (Luolajan-Mikkola, 2024).

Joshi (2024) mengkaji penerapan *Infrastructure as Code* (IaC) dengan mengombinasikan Terraform dan YAML guna meningkatkan efisiensi pengelolaan infrastruktur *cloud*. Integrasi YAML sebagai format konfigurasi deklaratif bersama Terraform difokuskan untuk menyederhanakan proses *provisioning*, meningkatkan keterbacaan konfigurasi, serta mendukung skalabilitas dan kemudahan pemeliharaan infrastruktur. Pada lingkungan *Microsoft Azure*, implementasi dilakukan dengan memanfaatkan modul Terraform, pengelolaan *state*, serta integrasi pipa CI/CD dan sistem *version control* berbasis *GitHub*. Melalui metodologi yang mencakup kajian literatur, studi kasus implementasi, dan evaluasi efisiensi, hasil penelitian membuktikan bahwa pendekatan ini mampu mempercepat penyediaan sumber daya, memangkas kompleksitas konfigurasi, serta memperkuat konsistensi dan kolaborasi. Kendati penggunaan YAML berhasil menyederhanakan penulisan sintaks, proses *provisioning* dalam kerangka kerja tersebut tetap mensyaratkan intervensi pengguna untuk memodifikasi berkas konfigurasi secara manual. Di samping itu, sistem ini belum menyediakan mekanisme isolasi keamanan berbasis *Role-Based Access Control* (RBAC) yang krusial untuk mencegah konflik dan potensi peretasan pada lingkungan infrastruktur privat berbasis *multi-tenant* .

Pembuktian efisiensi *Infrastructure as Code* (IaC) menggunakan Terraform pada lingkungan *public cloud* (AWS) dipaparkan oleh Muniruzzaman (2022). Hasil kajiannya mengonfirmasi bahwa otomatisasi *provisioning* dan *deployment* aplikasi berbasis Terraform mampu menekan kerumitan sistem, mengurangi proses operasional yang repetitif, serta meminimalkan risiko *human error* melalui tata kelola sumber daya yang konsisten. Meskipun demikian, kerangka kerja yang dirancang memiliki keterikatan tinggi terhadap ekosistem spesifik AWS (*vendor lock-in*), sehingga metodenya tidak dapat diimplementasikan secara langsung pada infrastruktur *cloud* privat lokal (*on-premise*). Dari aspek operasional, solusi ini masih berorientasi pada pengguna teknis karena proses penyediaannya pada umumnya bergantung pada keahlian penulisan berkas konfigurasi serta interaksi antarmuka baris perintah (CLI). Ketiadaan antarmuka visual membuat solusi otomatisasi tersebut terbatas pada kalangan pengembang (*DevOps*) dan belum

memfasilitasi kebutuhan *self-service* bagi pengguna nonteknis. Di samping itu, penelitian tersebut belum mengakomodasi mekanisme keamanan terpadu, seperti otomatisasi penanaman kunci autentikasi (*SSH Key*) saat peluncuran server baru.

Kajian mendalam mengenai fleksibilitas Terraform sebagai instrumen *Infrastructure as Code* (IaC) pada ekosistem *cloud* maupun *on-premise* dikemukakan oleh Krishnan (2024). Studi ini menguraikan keunggulan pendekatan deklaratif berbasis *HashiCorp Configuration Language* (HCL) yang memungkinkan pendefinisian kondisi akhir infrastruktur secara terstruktur dan konsisten. Di samping itu, dipaparkan alur kerja standar Terraform yang mencakup tahapan *write*, *plan*, dan *apply*, serta peran *provider plugin* dalam mengabstraksi interaksi ke berbagai penyedia infrastruktur. Evaluasi penelitian ini juga menjangkau ragam lingkungan eksekusi, seperti CLI, *Terraform Cloud*, *Terraform Enterprise*, hingga *Cloud Development Kit* (CDK), serta mengomparasikan Terraform dengan peranti sejenis seperti Chef, Puppet, CloudFormation, dan Heat. Kendati berfokus pada efisiensi otomatisasi *provisioning* layanan *cloud*, implementasi kerangka kerja yang diusulkan pada umumnya masih bergantung pada eksekusi skrip antarmuka baris perintah (CLI) yang kompleks. Penelitian tersebut juga belum mengintegrasikan mekanisme keamanan preventif, seperti otomatisasi injeksi *SSH Key* untuk tata kelola pengguna saat proses *provisioning* berlangsung

Shinde (2025) mengevaluasi penerapan *Infrastructure as Code* (IaC) menggunakan Terraform dalam mendukung otomatisasi pengelolaan infrastruktur pada layanan berbasis *cloud*. Studi ini menyoroti Terraform sebagai peranti IaC bersifat *open-source* dengan sintaks deklaratif yang mendukung implementasi *multi-cloud* melalui mekanisme *provider* terintegrasi. Pembahasan mencakup arsitektur Terraform, alur kerja utama yang melibatkan proses *plan* dan *apply*, serta penerapannya dalam praktik *DevOps* dan integrasi *pipeline* CI/CD. Selain itu, disajikan hasil evaluasi eksperimental yang membuktikan keunggulan Terraform dari aspek kecepatan *provisioning*, konsistensi konfigurasi infrastruktur, dan efektivitas kolaborasi tim dibandingkan alat IaC sejenis. Peneliti juga mengidentifikasi berbagai tantangan adopsi, seperti pengelolaan *state file*, kompleksitas modul dan dependensi, hingga isu keamanan dan standardisasi guna

memberikan gambaran menyeluruh mengenai pengelolaan infrastruktur *cloud* modern. Kendati arsitektur *provisioning* yang dirancang dinilai optimal untuk memfasilitasi alur kerja *DevOps engineer*, solusinya masih terlampau kompleks bagi pengguna nonteknis karena belum menghadirkan mekanisme layanan mandiri (*self-service*) yang mampu mengabstraksi kerumitan kode IaC dari lapisan operasional.

Penggunaan Terraform sebagai peranti *Infrastructure as Code* (IaC) untuk merancang solusi *disaster recovery* pada aplikasi berbasis *cloud-native* di lingkungan *multi-cloud* dieksplorasi oleh Tong (2022). Arsitektur aktif-pasif diterapkan dalam studi tersebut dengan mengintegrasikan *Amazon Web Services* (AWS) sebagai infrastruktur utama dan *Microsoft Azure* sebagai lingkungan cadangan. Berdasarkan uji eksperimental, otomatisasi IaC terbukti mampu memangkas waktu pemulihan hingga 75-85% sekaligus menghasilkan penyediaan infrastruktur lintas platform yang jauh lebih konsisten dan terstandar daripada metode manual. Kendati demikian, skema otomasi yang dirancang ditujukan secara spesifik untuk alur kerja praktisi *DevOps*, sehingga tingkat kompleksitas operasionalnya tergolong tinggi bagi pengguna nonteknis. Ketiadaan lapisan antarmuka visual untuk menyembunyikan kerumitan kode konfigurasi menyebabkan sistem ini belum mampu menyajikan pengalaman *self-service* bagi pengguna akhir. Di samping itu, karena orientasi utamanya berfokus pada pemulihan ketersediaan sistem, aspek tata kelola keamanan tidak tercakup dalam cakupan penelitian tersebut.

Otomatisasi penyediaan *Infrastructure as a Service* (IaaS) melalui kerangka kerja Terraform terbukti mampu mengabstraksi beragam layanan infrastruktur berbasis modul *provider*, sehingga efektif mencegah risiko keterikatan pada satu penyedia *cloud* (*vendor lock-in*). Kendati menawarkan keunggulan dari segi fleksibilitas arsitektur, tinjauan kritis menunjukkan adanya batasan implementasi pada aspek aksesibilitas pengguna akhir. Model operasional yang dirumuskan tergolong *engineer-centric*, di mana interaksi sistem wajib dioperasikan melalui antarmuka baris perintah (CLI) atau diintegrasikan lewat bahasa pemrograman menggunakan *Cloud Development Kit* (CDK). Ketiadaan lapisan antarmuka grafis (*dashboard* GUI) membuat kerangka kerja ini tidak aplikatif untuk memfasilitasi

skenario *self-service* bagi pengguna nonteknis. Selain itu, akibat fokus penelitian yang eksklusif pada penyediaan infrastruktur dasar, aspek tata kelola keamanan operasional untuk lingkungan infrastruktur berbagi (*multi-tenant*) tidak tercakup. Sistem ini belum mengakomodasi integrasi *Role-Based Access Control* (RBAC) maupun otomatisasi penanaman kunci autentikasi *SSH Key*, yang padahal menjadi prasyarat fundamental dalam mengamankan pemanfaatan infrastruktur virtualisasi privat seperti Proxmox VE (Howard, 2022).

Pemanfaatan *Infrastructure as Code* (IaC) menggunakan Terraform dalam skenario penyediaan infrastruktur lintas platform (*multi-cloud*) mencakup integrasi penyedia layanan *public cloud* utama seperti AWS, Microsoft Azure, dan Google Cloud Platform (GCP). Implementasi ini menyoroti keunggulan arsitektur berbasis *provider*, konfigurasi modular, dan manajemen *state* yang memungkinkan orkestrasi skala besar guna meningkatkan fleksibilitas, membangun redundansi ketersediaan, serta mencegah keterikatan pada satu vendor (*vendor lock-in*). Kendati terbukti andal pada ekosistem *cloud* yang kompleks, kerangka kerja yang dibangun dirancang secara eksklusif untuk kebutuhan korporasi berskala masif (*enterprise*). Tingkat kerumitan tersebut menjadikannya kurang aplikatif jika diterapkan untuk efisiensi operasional harian pada lingkungan *cloud* privat lokal (*on-premise*) seperti Proxmox VE. Terlebih lagi, pengelolaannya masih dioperasikan secara murni oleh *Cloud Engineer* melalui kode deklaratif dan antarmuka baris perintah (CLI), tanpa mengakomodasi aspek fundamental tata kelola pengguna seperti manajemen kuota sumber daya maupun otomatisasi injeksi kunci autentikasi *SSH Key* (Kyadasu dkk., 2024).

Penelitian yang dilakukan oleh Jayaram dkk. (2025) mengeksplorasi pemanfaatan *Infrastructure as Code* (IaC) menggunakan alat Terraform untuk mengakselerasi pengembangan infrastruktur komputasi awan. Studi ini mengintegrasikan Terraform dengan ekosistem AWS, sistem kontrol versi (Git), serta alat otomatisasi *pipeline* (Jenkins) guna membangun alur kerja penerapan (*deployment*) yang terstruktur. Hasil pengujian dalam studi tersebut membuktikan bahwa kerangka kerja otomasi ini secara signifikan mampu meningkatkan kecepatan penyediaan infrastruktur, menjamin konsistensi sumber daya komputasi, dan mendongkrak efisiensi operasional secara keseluruhan. Berdasarkan literatur

ini memperlihatkan bahwa arsitektur yang diusulkan sangat berorientasi pada pengembang perangkat lunak (*developer-centric*). Alur operasionalnya mensyaratkan pengguna untuk memiliki pemahaman teknis mendalam terkait siklus *Continuous Integration/Continuous Deployment* (CI/CD) serta keahlian menulis kode konfigurasi. Ketiadaan lapisan antarmuka visual (*dashboard GUI*) menjadikan solusi otomasi ini terisolasi bagi praktisi teknis dan luput memfasilitasi kebutuhan layanan mandiri (*self-service*) bagi pengguna awam yang tidak memiliki latar belakang pemrograman. Selain itu, karena penelitian tersebut berfokus secara eksklusif pada lingkungan awan publik komersial (AWS), kajian ini belum menyentuh tantangan tata kelola operasional dan keamanan pada infrastruktur awan privat. Aspek esensial seperti pembatasan hak akses multipengguna (RBAC) maupun otomatisasi penanaman otentikasi *SSH Key*, yang menjadi prasyarat mutlak pada platform virtualisasi lokal seperti Proxmox VE, tidak terakomodasi dalam kerangka kerja tersebut.

Tabel 2.1 merangkum dari penelitian terdahulu yang telah menerapkan IaC untuk *provisioning virtual machine* pada infrastruktur virtualisasi.

Tabel 2.1 Penelitian Terkait

No	Judul	Penulis	Tahun	Teknologi	Hasil Utama	Celah Penelitian
1	Ink: An Infrastruktur as Code Tool for Arbitrary Compute Environments	Luolajan-Mikkola	2024	OpenTofu, Modular Ink	Mengembangkan prototipe *Infrastructure as Code* (Ink) untuk lingkungan komputasi arbitrer yang belum memiliki dukungan IaC bawaan	Meskipun menawarkan modularitas, interaksi sistem masih sepenuhnya berbasis kode/CLI tanpa antarmuka visual (*dashboard*), sehingga belum mendukung layanan mandiri (*self-service*) yang ramah pengguna awam maupun fitur keamanan kredensial otomatis.
2	Simpliying Infrastructure Management with Terraform and YAML Configuration	Joshi	2024	Terraform, YAML, Microsoft Azure	Kerangka kerja IaC berbasis Terraform dan YAML untuk menyederhanakan manajemen konfigurasi infrastruktur *cloud* Azure	Penggunaan YAML berhasil menyederhanakan sintaks, namun proses *provisioning* tetap mensyaratkan modifikasi berkas konfigurasi manual. Sistem belum menyediakan isolasi keamanan berbasis peran (RBAC) untuk penggunaan bersama.
3	Infrastructure Automation Using *Infrastructure as*	Moniruzzaman	2024	Terraform, Ansible, AWS	Model otomatisasi infrastruktur *Public Cloud* (AWS) yang mengintegrasikan Terraform dan Ansible untuk konfigurasi server otomatis.	Fokus penelitian terpaku pada lingkungan Public Cloud (AWS) yang berbayar. Belum menawarkan solusi efisiensi untuk *Private Cloud*, serta tidak

No	Judul	Penulis	Tahun	Teknologi	Hasil Utama	Celah Penelitian
	Code					memiliki lapisan antarmuka grafis (GUI) untuk pengguna non-teknis.
4	Infrastructure Automation and Management Using *Infrastructure as Code*	Krishnan, P	2022	*Terraform, Ansible, AWS.*	Kerangka kerja otomatisasi yang menekankan standardisasi konfigurasi dan manajemen sumber daya *cloud.*	Implementasi masih bergantung pada skrip baris perintah (CLI) yang kompleks. Belum mengintegrasikan mekanisme keamanan preventif seperti injeksi SSH Key otomatis pada saat *provisioning*.
5	Infrastructure *provisioning* and Automation Using *Infrastructure as Code*	Shinde, S	2025	Terraform, Ansible, AWS	Arsitektur *provisioning* otomatis yang terintegrasi dengan *pipeline* CI/CD untuk meningkatkan keandalan *deployment*	Sangat optimal untuk alur kerja DevOps engineer, namun terlalu kompleks untuk pengguna akhir (end-user). Belum menyediakan mekanisme self-service yang memisahkan kompleksitas kode dari operasional.
6	Automated Cloud Infrastructure Deployment Using Infrastructure as Cod	Wei Tong	2022	Terraform, AWS, Azure	Solusi *Disaster Recovery* otomatis untuk aplikasi *cloud-native* pada lingkungan *multi-cloud*	Fokus pada skenario pemulihan bencana (*Disaster Recovery*) di *Public Cloud* untuk teknisi *DevOps*. Belum menyediakan *dashboard self-service* bagi pengguna nonteknis, serta luput mengintegrasikan tata kelola keamanan harian (injeksi *SSH*

No	Judul	Penulis	Tahun	Teknologi	Hasil Utama	Celah Penelitian
						Key dan manajemen kuota) pada lingkungan *Private Cloud*
7	Terraform Automating Infrastructure as a Service	Howard, M	2022	Terraform, Multi Cloud	Kajian komprehensif alur kerja Terraform (*write-plan-apply*) untuk *provisioning* lintas platform.	Terbatas pada fungsionalitas dasar IaC yang sangat berorientasi pada teknisi (berbasis CLI/Kode). Belum menawarkan antarmuka visual (*dashboard*) untuk memfasilitasi kemandirian pengguna awam, serta luput membahas integrasi keamanan multipengguna (injeksi *SSH Key* dan RBAC) untuk infrastruktur *Private Cloud*
8	Exploring *Infrastructure as Code* Using Terraform in Multi-Cloud Deployments	Kyadasu, dkk	2024	Terraform, AWS, Google Cloud Platform, Microsoft Azure	Penerapan IaC untuk menyatukan manajemen konfigurasi di berbagai penyedia layanan *cloud* (*Multi-Cloud*)	Fokus pada orkestrasi *multi-cloud* (AWS, Azure, GCP) berskala korporasi yang dioperasikan murni melalui antarmuka kode oleh teknisi khusus. Belum menyediakan antarmuka visual (*dashboard*) untuk memfasilitasi kemandirian pengguna awam, serta luput mengintegrasikan tata kelola keamanan spesifik pengguna (manajemen kuota dan injeksi

No	Judul	Penulis	Tahun	Teknologi	Hasil Utama	Celah Penelitian
						SSH Key) pada lingkungan *Private Cloud* lokal
9	Accelerated Cloud Infrastructure Development Using Terraform	Vivekananda Jayaram, dkk	2024	Terraform, AWS, Jenkins	Akselerasi pengembangan infrastruktur *cloud* menggunakan integrasi Terraform dan alat CI/CD Jenkins.	Target pengguna adalah pengembang teknis (*developers*) yang terbiasa dengan kode. Tidak menyediakan fitur kemandirian (*self-service*) bagi pengguna non-teknis yang membutuhkan akses cepat tanpa *coding*
10	Penelitian Ini (Usulan	Harumin	2026	Proxmox, Terraform, Python	*Dashboard self-service* yang memungkinkan pengguna non-teknis melakukan *provisioning* VM secara mandiri, aman, dan terstandarisasi.	Mengisi celah dengan menghadirkan antarmuka grafis (GUI) pada lingkungan Private Cloud (Proxmox), serta menerapkan keamanan preventif melalui injeksi SSH Key otomatis dan isolasi hak akses (RBAC) untuk menjamin efisiensi dan keamanan.

2.2 Landasan Teori

Pengembangan portal layanan mandiri (*self-service*) laboratorium virtual berlandaskan pada integrasi konsep *Infrastructure as Code* (IaC) dan otomasi virtualisasi. Bagian ini menyajikan landasan teoritis mengenai Terraform, Proxmox VE, serta mekanisme *provisioning* mesin virtual secara dinamis, yang diperkuat dengan tinjauan terhadap penelitian terdahulu yang relevan.

2.2.1 Transformasi Digital dalam Layanan Teknologi Informasi

Transformasi digital merupakan proses integrasi teknologi digital ke dalam proses bisnis dan operasional organisasi untuk meningkatkan efisiensi, kualitas layanan, serta kemampuan adaptasi terhadap perubahan lingkungan dan kebutuhan pengguna (A. Ramirez dkk., 2025). Dalam konteks layanan teknologi informasi, transformasi digital mendorong pergeseran dari pengelolaan infrastruktur yang bersifat manual menuju layanan TI yang terotomasi, responsif, dan berorientasi pada pengguna (Verhoef dkk., 2021). Seiring meningkatnya kebutuhan layanan digital, institusi dituntut untuk menyediakan infrastruktur TI yang fleksibel, dapat disediakan dengan cepat, dan dapat dikelola secara mandiri oleh pengguna dengan tetap memperhatikan aspek keamanan dan tata kelola (Vial, 2019).

2.2.2 Virtualisasi dan Infrastruktur Virtualisasi Privat

Virtualisasi merupakan teknologi yang memungkinkan pemisahan sumber daya fisik menjadi beberapa lingkungan virtual yang berjalan secara independen. Teknologi ini menjadi fondasi utama pengelolaan infrastruktur TI modern karena mampu meningkatkan efisiensi pemanfaatan sumber daya, fleksibilitas, serta skalabilitas layanan (Gill dkk., 2022). Infrastruktur virtualisasi privat memberikan tingkat kendali yang lebih tinggi kepada institusi terhadap sumber daya dan data yang dikelola, sehingga banyak dimanfaatkan untuk mendukung layanan internal yang bersifat strategis dan sensitif (Alouffi dkk., 2021). Namun, seiring meningkatnya beban kerja dan jumlah pengguna, pengelolaan infrastruktur virtualisasi privat secara manual menghadapi tantangan berupa kompleksitas pengelolaan infrastruktur, inkonsistensi konfigurasi (*configuration drift*), serta peningkatan risiko kesalahan operasional *(human error)* (Rahman dkk., 2020).

2.2.3 Provisioning Virtual machine

Provisioning VM merupakan proses penyediaan dan konfigurasi mesin virtual sesuai dengan spesifikasi yang telah ditentukan, mencakup alokasi sumber daya komputasi, sistem operasi, serta konfigurasi jaringan (Park dkk., 2025). Pada banyak lingkungan virtualisasi privat, proses *provisioning* masih dilakukan secara manual oleh administrator sistem. Pendekatan ini berpotensi menyebabkan waktu *provisioning* yang lebih lama, ketidakkonsistenan konfigurasi antar mesin virtual, serta meningkatnya risiko kesalahan manusia *(human error)* (Gill dkk., 2022) (Rahman dkk., 2020). Kondisi tersebut menjadi hambatan dalam mendukung layanan TI yang adaptif dan berkelanjutan, sehingga diperlukan mekanisme *provisioning* yang terstandarisasi dan terotomasi.

2.2.4 Konsep Layanan Self-Service pada Infrastruktur TI

Dashboard self-service pada infrastruktur TI merupakan pendekatan yang memungkinkan pengguna untuk melakukan permintaan layanan secara mandiri melalui antarmuka terpusat tanpa keterlibatan langsung administrator pada setiap proses (Mell & Grance, 2011) (Nastic, 2024). Otomatisasi penyediaan infrastruktur menggunakan pendekatan *Infrastructure as Code (IaC)*, seperti Terraform, memungkinkan proses *provisioning* sumber daya komputasi, konfigurasi sistem, dan pengaturan jaringan dilakukan secara konsisten dan terstandarisasi berdasarkan definisi kode. Pendekatan ini bertujuan untuk meningkatkan efisiensi operasional, mempercepat waktu *provisioning* infrastruktur, serta mengurangi beban kerja tim pengelola infrastruktur (Howard, 2022). Dalam konteks virtualisasi, *dashboard self-service* berperan sebagai penghubung antara pengguna dan sistem otomasi *provisioning*, sehingga pengguna nonteknis tetap dapat mengakses layanan infrastruktur sesuai dengan kebijakan dan hak akses yang telah ditetapkan.

2.2.5 Infrastructure as Code

Infrastructure as Code (IaC) merupakan pendekatan pengelolaan infrastruktur TI dengan mendefinisikan konfigurasi infrastruktur dalam bentuk kode yang dapat dijalankan secara otomatis. Pendekatan ini mengadopsi prinsip rekayasa perangkat lunak seperti otomasi, pengendalian versi, dan penggunaan ulang. Dengan IaC, proses *provisioning* infrastruktur dapat dilakukan secara konsisten, tervalidasi, dan dapat direplikasi, sehingga mengurangi ketergantungan

pada konfigurasi manual yang rentan terhadap kesalahan. IaC menjadi fondasi utama dalam mendukung otomasi *provisioning* pada lingkungan virtualisasi *privat* (Kyadasu dkk., 2024).

2.2.6 Terraform sebagai Implementasi Infrastructure as Code

Terraform merupakan salah satu alat IAC yang menggunakan pendekatan deklaratif dalam mendefinisikan kondisi akhir infrastruktur yang diinginkan. Pengguna hanya perlu mendeskripsikan sumber daya yang dibutuhkan, sementara proses penyediaan dan pengelolaan perubahan dilakukan secara otomatis oleh sistem. Terraform mendukung konsistensi konfigurasi, otomasi *provisioning*, serta integrasi dengan berbagai platform virtualisasi dan *cloud*, sehingga sesuai digunakan sebagai platform otomasi *provisioning* dalam sistem *dashboard self-service* (Hashicorp, 2024).

2.2.7 Infrasruktur Virtual berbasis Proxmox Virtual Environment

Proxmox Virtual Environment (PVE) merupakan platform virtualisasi *open-source* yang mendukung pengelolaan mesin virtual dan *container* dalam lingkungan infrastruktur privat. PVE menyediakan mekanisme manajemen terpusat melalui antarmuka web serta *Application Programming Interface* (API) yang mendukung otomasi pengelolaan sumber daya virtual (Proxmox Server Solutions GmbH, 2025). Dalam penelitian ini, Proxmox VE digunakan sebagai lingkungan virtualisasi privat yang menjadi objek implementasi otomasi *provisioning* berbasis *Infrastructure as Code* dan *dashboard self-service*.

2.2.8 Keamanan Infrstruktur Virtualisasi

Keamanan infrastruktur virtualisasi merupakan aspek krusial dalam pengelolaan layanan TI, khususnya pada lingkungan multipengguna. Ancaman keamanan siber, seperti serangan berbasis kredensial, dapat mengeksploitasi kelemahan mekanisme autentikasi dan kontrol akses, sehingga menyebabkan akses tidak sah terhadap sumber daya virtual (NIST, 2020). Oleh karena itu, penerapan mekanisme keamanan yang mencakup kontrol akses, autentikasi yang kuat, serta

prinsip *least privilege* menjadi kebutuhan utama dalam sistem *provisioning* VM berbasis *self-service.*

2.2.9 Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC) merupakan metode pengelolaan hak akses yang memberikan izin kepada pengguna berdasarkan peran yang dimilikinya dalam sistem. RBAC memungkinkan pengelolaan akses yang terstruktur dan konsisten, serta mendukung penerapan prinsip *least privilege* (Singh dkk., 2024). Dalam sistem *dashboard self-service*, RBAC berperan penting dalam memastikan bahwa pengguna hanya memperoleh akses terhadap fungsi dan sumber daya virtual sesuai dengan perannya, sehingga meminimalkan risiko penyalahgunaan hak akses.

2.2.10 Authentikasi berbasis SSH Public Key

Autentikasi berbasis *SSH Public key* merupakan mekanisme autentikasi yang menggunakan pasangan kunci kriptografi untuk memverifikasi identitas pengguna tanpa bergantung pada kata sandi. Dibandingkan dengan autentikasi berbasis kata sandi, metode ini memberikan tingkat keamanan yang lebih tinggi dan efektif dalam mengurangi risiko serangan *brute force* serta pencurian kredensial (Touil dkk., 2024). Selain meningkatkan keamanan, autentikasi berbasis *public key* juga mendukung otomasi *provisioning* karena memungkinkan proses akses sistem tanpa interaksi manual berbasis kata sandi.