Jurnal Ilmu Siber dan Teknologi Digital (JISTED)
ISSN: 2986-7312, Vol 5, No 1, 2026, 49-68 https://doi.org/10.61401/jisted.v5.n1.p49-68.2026

ExoVirt: Self-Service Virtual Machine Provisioning on Proxmox Virtual Environment Using Infrastructure as Code

Ardi Maryanto[1*], Zulkifli Zulkifli[2], Panji Bintoro[3], Agus Wantoro[4]
Universitas Aisyah Pringsewu, Pringsewu, Indonesia[1,2,3,4]
ardyxmr@gmail.com[1*], zulkifli@aisyahuniversity.ac.id[2], panjibintoro09@aisyahuniversity.ac.id[3], aguswantoro@aisyahuniversity.ac.id[4]

Article History
Received on 24 August 2026
1st Revision on 27 August 2026
2nd Revision on 01 September 2026
Accepted on 25 September 2026

Abstract
Purpose: This study designs, builds, and evaluates ExoVirt, an open-source web platform that abstracts Infrastructure as Code so that non-expert users can provision virtual machines on their own atop Proxmox Virtual Environment.
Research Methodology: This study applied the Design Science Research Methodology. The artifact was built with React, Laravel, PostgreSQL, Redis, Terraform, and Ansible, and then evaluated through functional (black box), efficiency, consistency, human-error, and usability (System Usability Scale) tests against native Proxmox VE.
Results: The SUS score was 92.19 for ExoVirt and 37.81 for Proxmox VE, confirmed by a paired sample t-test ($t(7) = 6.982$; $p = 0.000215$) and a Wilcoxon signed-rank test ($W = 0$; $p = 0.0078$). The number of steps fell from 23 to 10 (56.52%), all 40 scenarios were passed, configuration consistency reached 100% without drift, configuration error was zero, and provisioning time fell 27.83%.
Conclusions: Abstracting Infrastructure as Code behind a self-service interface allows non-expert users to provision virtual machines themselves, while access control, approval, and audit logging remain intact.
Limitations: The researcher also served as the developer during measurement, the SUS sample held eight respondents, and the consistency design narrowed the difference to a single parameter.
Contributions: This study offers a reference design and empirical evidence for infrastructure governance and automation.

Keywords: *Infrastructure as Code, Infrastructure Governance, Proxmox VE, Self-Service Provisioning, System Usability Scale*

How to Cite: Maryanto, A., Zulkifli, Z., Bintoro, P., & Wantoro, A. (2026). ExoVirt: Self-Service Virtual Machine Provisioning on Proxmox Virtual Environment Using Infrastructure as Code. *Jurnal Ilmu Siber dan Teknologi Digital*, 5(1), 49-68.

1. Introduction

Organizations adopt virtualization to reduce hardware costs and speed up the provisioning of computing environments. Virtual machines optimize physical resources under one central platform (Marzuki et al., 2023), and Giallorenzo et al. (2021) show that virtual-machine virtualization remains a foundation of computing services even as containers grow, because many production workloads require full isolation. Many organizations still provision these machines by hand through the hypervisor interface, where an administrator creates the machine, sets compute

resources, configures networking, assigns storage, and prepares the operating system one machine at a time.

This manual process became a burden under load. Service-request data from a banking institution running a VMware-based infrastructure showed the following pattern: two administrators handled a fleet of 700 to 2,400 virtual machines. Between March and April, the institution logged 120 provisioning tickets, climbing from 30 in March to 90 in April as a new project started, and capacity increased with new-machine creation driving about 72 percent of them. One machine takes 5 to 10 minutes to build for a standard configuration and 10 to 20 minutes when it needs extra disk and packages; thus, a spike in requests turned the two administrators into a bottleneck. The same records exposed a second cost: across the 120 operations, 16 carried configuration errors, about 13.3 percent, concentrated on machines created or modified close together, where repeating identical steps for every request opened room for mistakes.

Infrastructure as Code (IaC) can cut down both the time and the error, yet it raises a new barrier. Terraform provisions virtual machines, networks, and storage from configuration files, and Ansible carries out advanced configuration and post-provisioning hardening, but both tools demand fluency in configuration syntax, the command line, Terraform state, Ansible inventory, and platform details such as node names, networks, and storage pools. That fluency keeps IaC inside the administrator team, so the users who need machines still wait on them. Opening provisioning to users without control creates its own risk: unchecked resource use, non-standard configuration, and activity that no one can trace. Approval, role-based access control, lifecycle management, and audit logging have to stay in place, and because operating-system misconfiguration and the security smells that IaC files often carry both widen the attack surface, standardized hardening stays necessary (Rahman & Williams, 2021). The pressing need is a self-service layer that non-experts can operate while governance and security stay intact.

IaC underpins this direction by describing infrastructure as machine-readable code that provisions resources automatically, repeatably, and consistently (Rahman et al., 2019). Terraform and Ansible form the common toolchain for automation, and Proxmox VE offers a fitting foundation to run it: open source with no license fee, an open API well served by the Terraform provider, and a lighter footprint than OpenStack. Educational institutions already run Proxmox VE as campus server infrastructure and as an academic cloud (Kardha et al., 2020; Oleksiuk & Oleksiuk, 2021). Prior research has automated pieces of this stack, from server configuration to virtual networking to security hardening.

These efforts remain separate. Studies automate server configuration, virtual networking, or hardening on their own, while work that unites self-service requests, an approval workflow, resource governance, audit logging, Terraform orchestration, and Ansible hardening in one web application remains scarce. Platforms that reach comparable breadth are usually licensed enterprise products with high cost, and open-source options that fold all of these needs into a single platform are rare.

The specific shape of that gap has been shown in prior work. Hariyadi and Marzuki (2020) and Khumaidi (2021) automated server and account configuration with Ansible but offered no self-service portal or approval path. Marzuki et al. (2023) automated virtual-network configuration on Proxmox rather than machine provisioning. Nelmiawati et al. (2025) scripted provisioning and hardening with Terraform and Ansible yet left the workflow without a portal or role-based access control. Nabila et al. (2025) compared the two tools instead of building a system, and Sandobalín et al. (2020) evaluated tool effectiveness rather than an end-user portal. Fadhudin and Ramayanti (2026) automated service configuration without self-service provisioning, approval, or RBAC. Each result advances one layer of automation; none binds the request, approval, governance, orchestration, and hardening layers into one interface a non-expert can operate.

This study addresses this gap with ExoVirt, a web application built on Proxmox VE that hides Terraform and Ansible behind an abstraction layer so that non-expert users can order, create, and manage virtual machines on their own without writing technical commands. The novelty of this work lies in uniting self-service provisioning, an approval workflow, role-based access control, audit logging, Terraform orchestration, and Ansible hardening in a single open-source web application on Proxmox VE, a combination prior work has not delivered. The work pursues four objectives: (1) design a self-service web application that abstracts IaC complexity for non-expert users; (2) embed governance through role-based access control, an approval workflow, lifecycle management, and audit logging; (3) evaluate operational efficiency, configuration consistency, and human error against the manual process; and (4) evaluate usability with the System Usability Scale (SUS) against the native Proxmox VE interface. The study contributes a reference design and empirical evidence for infrastructure governance and automation, and a practical path that lets non-expert users provision virtual machines without surrendering access control, approval, or audit capability.

2. Literature Review and Hypotheses Development

2.1 Prior Studies

Research on infrastructure automation has advanced along several directions. Rahman et al. (2019) mapped IaC research systematically and found that most studies focus on tools, testing, and infrastructure code quality, while the end-user experience is rarely addressed. Kumara et al. (2021) summarized good and bad practices of writing infrastructure code, such as the use of version control, modularity, and testing, which confirms that mastering IaC demands software-engineering discipline. Sandobalín et al. (2020) contrasted model-driven with code-centric IaC tools and found that code-centric options such as Terraform require high syntactic precision. These three studies reinforce the argument that IaC is technically powerful but difficult for non-expert users to access.

At the local implementation level, Hariyadi and Marzuki (2020) used Ansible for virtual private server configuration management and reduced manual intervention. Khumaidi (2021) applied Ansible to automate server and user-account management. Marzuki et al. (2023) used Ansible to automate Open vSwitch networking on the Proxmox platform, which lowered human error. Nelmiawati et al. (2025) combined Terraform and Ansible for provisioning and hardening following NIST SP 800-123 so that security configuration became more consistent. Nabila et al. (2025) benchmarked Ansible against Terraform and reported Terraform as faster in provisioning execution. Fadhudin and Ramayanti (2026) cut the configuration time of one server from 27 to 8 minutes, or 70.4%, and across 14 servers cut an estimate of 378 to 9 minutes, or 97.6%. These data sharpen the efficiency gap between the manual and automated processes.

2.2 Theoretical Foundation

Cloud Computing and Self-Service, cloud computing allows users to access computing resources on demand over a network without directly managing the hardware, delivered through IaaS, PaaS, and SaaS service models (Nabila, Indrawati, & Davi, 2025). On-demand self-service is its core characteristic, and Qiu and Shen (2021) show that automated cloud resource provisioning speeds up demand fulfillment while preserving resource utilization. Yuan, Das, Ramesh, and Qiao (2023) examine how virtual resources can be provisioned with availability guarantees under cloud service agreements, while Pandey and Rawat (2024) and Radhika and Sudha (2021) developed automated and resource-efficient virtual-machine provisioning and management for data centers. This study applies the self-service principle as a web portal that lets users request and manage virtual machines on their own, while administrators set the policy. This line of self-service provisioning builds on broader research into on-demand resource provisioning and scheduling for IaaS clouds (Rajasekar, & Palanichamy, 2021; Samha, 2024; Shokri, Yousefi, & Ghalebsaz, 2021).

Virtualization and Proxmox VE, virtualization divides physical resources into several isolated environments through a hypervisor and thus improves hardware utilization (Marzuki, Kholid, Hariyadi, & Mardedi, 2023). Giallorenzo, Mauro, Poulsen, and Siroky (2021) measured the cost of

virtual-machine virtualization against containers and bare-metal and confirmed that virtual machines remain relevant when full isolation is needed. Proxmox VE is a Linux-based, open-source virtualization platform that administers virtual machines, virtual networks, and storage via a web console and an open API (Firmansyah, Winarno, & Pramono, 2019; Kardha, Pamungkas, & Setiawan, 2020). The API becomes the integration point for Terraform in this study. Beyond the platform itself, research on virtual-machine placement and scheduling keeps optimizing energy and resource use in data centers (Ajmera, & Kumar, 2024; Rjeib, & Kecskemeti, 2024; Singh, Singh, & Ajmera, 2026), and container orchestration extends the same automation to clustered workloads (Zhou, Georgiou, Pospieszny, Zhong, Zhou, Niethammer, Pejak, Marko, & Hoppe, 2021).

Infrastructure as Code, Terraform, and Ansible. IaC manages infrastructure through code files that can be stored, reviewed, and reused so that provisioning runs consistently and documented (Padur, 2021; Rahman, Mahdavi-Hezaveh, & Williams, 2019). Terraform applies a declarative approach to provision virtual machines, networks, and storage, while Ansible is agentless over SSH for post-provisioning configuration and hardening (Hariyadi, & Marzuki, 2020; Nelmiawati, Gunawan, Arif, Haikal, & Putra, 2025). The quality of IaC files draws attention: Rahman et al. (2019) found code properties that mark defective IaC files Palma, Nucci, Palomba, and Tamburri (2022) built a defect-prediction model for IaC files from product and process metrics, and Bessghaier, Ouni, Sayagh, Chouchen, and Mkaouer (2025) examined IaC code-review practice on OpenStack projects. These findings explain why writing IaC demands expertise and why abstracting it for non-expert users holds value. At the process level, Bernardo, da Costa, Kulesza, and Treude (2023) linked a continuous-integration service with change delivery time, which confirms automation as a determinant of release speed. Empirical DevOps research further maps adoption trends and performance-assurance practices in delivery pipelines(Rzig, Hassan, & Kessentini, 2022; Traini, 2022), while configuration-anomaly detection guards against drift in operational systems (Li, Cheng, Zhang, & Yu, 2023).

Access Control, Approval, and Audit Logging. Role-Based Access Control (RBAC) attaches permissions to roles rather than to individuals, and each user inherits the permissions of an assigned role, which keeps access-right management simpler and more consistent (Sandhu, Coyne, Feinstein, & Youman, 1996). Karatas and Akbulut (2018) mapped access-control mechanisms in cloud computing and placed RBAC as a widely applied model because it aligns with the least-privilege principle. Venčkauskas, Kukta, Grigaliūnas, and Brūzgienė (2023) applied token-based access control on a microservice architecture, and Logrippo (2025) examined data-flow security in RBAC. Recent work extends access control toward attribute-based and zero-trust models (Awan, Azad, Arshad, Waheed, & Sharif, 2023; Mao, Fu, Zhao, Yuan, Sun, & Zhao, 2025; Wang, Wang, Song, & Cheng, 2023). This body of work confirms access governance as a foundation still under active development. Approval workflow requires a request to pass review before execution, while audit logging records the actor, action, and time as evidence of accountability. These three mechanisms close the governance gap left open by uncontrolled self-service. Wang, Li, Zhang, and Tu (2025) designed a zero-trust dynamic access-control model for cloud computing.

Hardening strengthens operating system security by reducing the attack surface through standardized configuration. Nelmiawati et al. (2025) show that automated hardening with Ansible produces a more consistent security configuration than the manual approach, and Rahman and Williams (2021) stress that hardening remains necessary because infrastructure scripts themselves may hold gaps. Akbar and AlSanad (2025) identify the key enablers that make secure DevOps practices effective in organizations.

System Usability Scale (SUS uses ten statements rated on a five-point Likert scale and returns a 0-to-100 score for perceived ease of use (Brooke, 1996). Bangor, Kortum, and Miller (2008) evaluated SUS empirically across hundreds of studies and placed a score near 68 as the acceptability threshold, while Orfanou, Tselios, and Katsanos (2015) demonstrated its reliability on web-based systems. Lewis

(2018) summarizes that SUS stays reliable on small samples. Black box testing complements the evaluation by verifying that feature outputs match the specification (Pratama & Putra, 2022). Usability evaluation likewise remains an active method for assessing the user experience of purpose-built applications (Piki, Kasenides, & Paspallis, 2026). A recent study applied the SUS to evaluate an integrated web interface and reported good, acceptable usability for a user-centered prototype (Cecelya, Rahmadi, & Armin, 2024).

Design Science Research Methodology (DSRM designs and evaluates an artifact to solve a problem through six phases: problem identification, objective definition, design and development, demonstration, evaluation, and communication (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007). This methodology is suitable for research that produces and evaluates an artifact in the form of an application.

2.3 Research Gap and Hypothesis Development

Prior work proves that Terraform and Ansible improve efficiency and consistency; however, it focuses on the technical side of automation and has not addressed end-user ease of use or unified governance. No study has integrated self-service requests, approval workflows, role-based access control, audit logging, terraform provisioning, and Ansible hardening in a single web application on Proxmox VE. This study fills this gap and measures its impact on efficiency, consistency, human error, and usability against the native Proxmox VE interface. This comparison is framed into three hypotheses.

H_1: ExoVirt and the native Proxmox VE interface differ significantly in virtual machine provisioning efficiency

H_2: ExoVirt and the native Proxmox VE interface differ significantly in terms of configuration consistency

H_3: ExoVirt and the native Proxmox VE interface differ significantly in usability, as measured by the SUS score

3. Research Methodology

3.1 Research Type and Framework

This study applies the Design Science Research Methodology (DSRM) because it both produces and evaluates an artifact, a self-service web application, and not only describes a phenomenon (Peffers et al., 2007). The research runs through six phases, each with a defined activity and output.

Problem identification and motivation this study characterized the manual process through three data-collection procedures. Direct observation of the provisioning workflow on the running environment exposed the repeated configuration steps, from template selection to resource sizing, networking, and post-clone operating system setup, along with their time cost and inconsistency risk. Interviews with the infrastructure managers surfaced the need for structured self-service provisioning, an approval path, configuration standardization, and automation. A literature study of IaC, Terraform, Ansible, Proxmox VE, orchestration, and web-application security grounded the analysis and evaluation design.

Solution objective definition from this analysis, the study set four objectives: abstract IaC complexity for nonexpert users; embed governance through role-based access control, an approval workflow, lifecycle management, and audit logging; evaluate efficiency, configuration consistency, and human error against the manual process; and evaluate usability against the native Proxmox VE interface.

Design and development artifact development used the Prototyping method nested within DSRM because user needs matured incrementally and demanded fast feedback. Each cycle ran four stages: requirement identification, prototype construction, evaluation, and refinement, and repeated until the core features reached the expected function. The features were self-service provisioning, the approval

workflow, Terraform-based provisioning, and Ansible-based configuration and hardening. The design covered the system architecture, UML modeling, the database, and the abstraction-and-policy layer.

Demonstration the completed artifact ran end-to-end scenarios across a provisioning request, approval workflow, Terraform provisioning, Ansible configuration, and inventory management through the portal, which confirmed that the components operate as one integrated pipeline on Proxmox VE.

Evaluation the evaluation strategy follows the FEDS framework, which selects a method by purpose and risk (Venable et al., 2016). Four tests were run against the native Proxmox VE process: functional black-box testing across all features, efficiency testing on provisioning time and step count, configuration-consistency testing against the tier standard, and usability testing with the System Usability Scale. The study fixed the success indicators before measurement: 100% functional success, at least a 50% reduction in provisioning time alongside fewer steps, 100% configuration conformity without drift, and a SUS score of at least 68. Section 3.4 details the protocol and analysis.

Communication this study communicates its results to two audiences. For the academic audience, the first author's undergraduate thesis and this article present the problem, artifact design, evaluation evidence, and novelty of uniting self-service provisioning with governance on Proxmox VE. For the practitioner audience, ExoVirt runs in the production environment of the host organization, and its source is available from the first author on request, with a public release planned. Figure 1 show research framework based on DSRM with nested Prototyping.

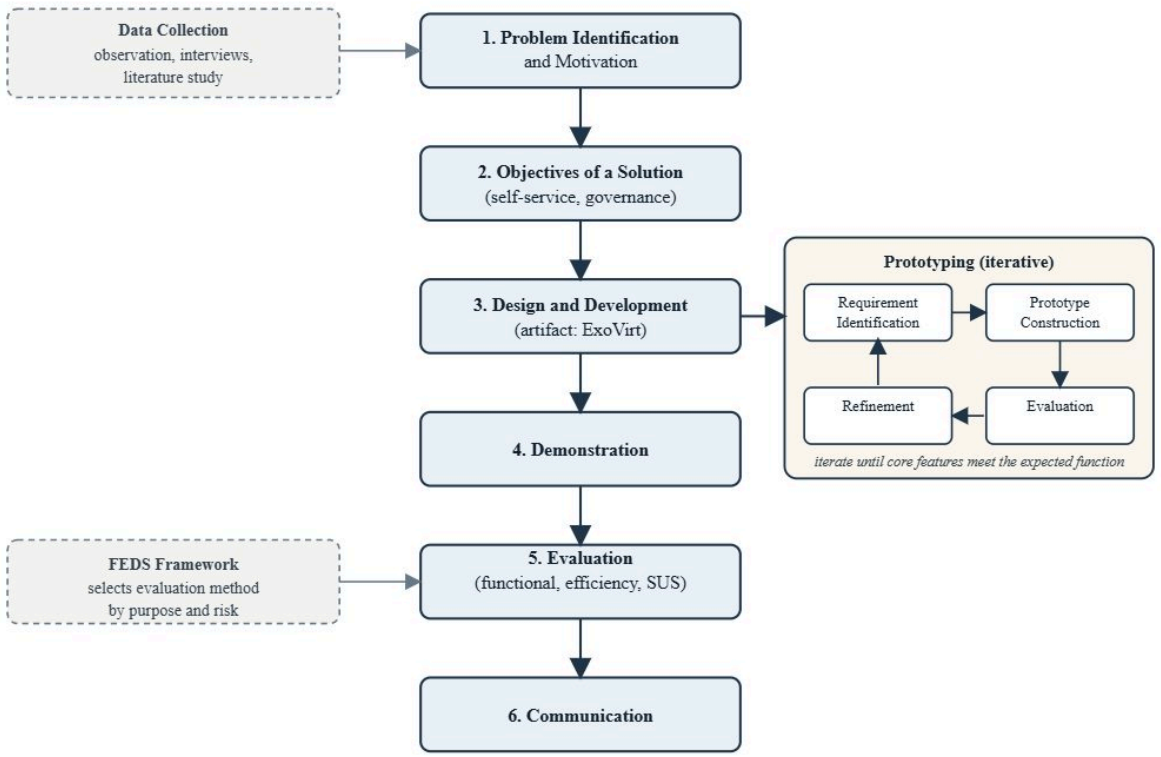


Figure 1. Research framework based on DSRM with nested Prototyping

3.2 System Architecture

The user interface is built with React.js. User requests are processed by a Laravel backend that handles authentication, authorization, approval workflow, inventory management, and integration with Terraform and Ansible. PostgreSQL stores all application data, Redis handles caching and the job

queue, and Laravel Reverb provides real-time WebSocket-based communication. Terraform runs provisioning through the Proxmox VE API, and then Ansible configures and hardens the guest operating system per policy. The system architecture appears in Figure 2.

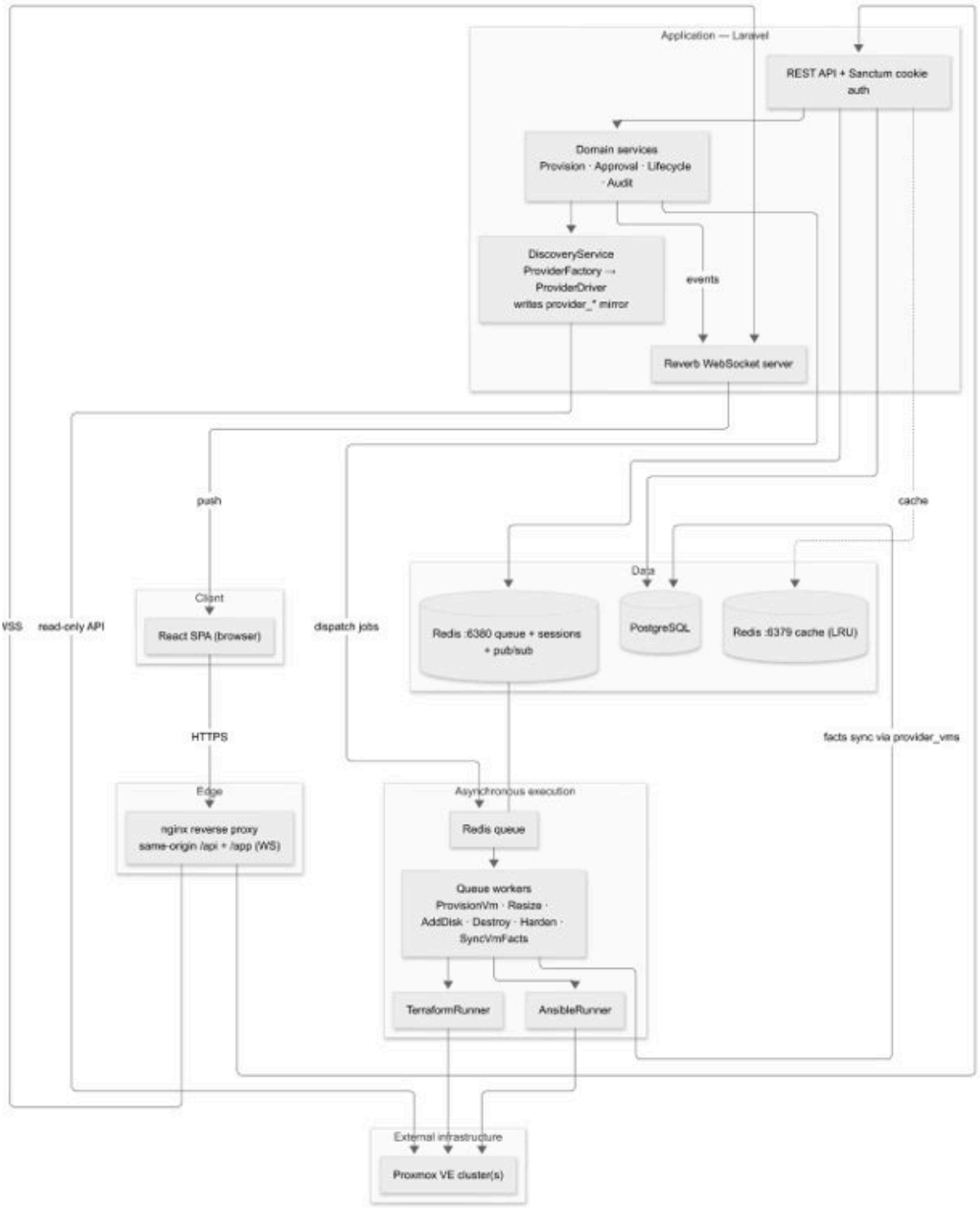


Figure 2. ExoVirt system architecture

The system separates infrastructure-resource management from user services through an abstraction and policy layer. A discovery mechanism registers resources on the provider, the discovery result is published as a service catalog, then policy in the form of environment, resource tier, and approval workflow controls provisioning in a standardized way. The system applies three roles, namely User

(Requestor), Approver (Manager), and Admin, with a generalization relationship, so the Admin access scope covers the Approver and the Approver covers the User.

3.3 Variables and Instruments

Four variables were evaluated: efficiency, configuration consistency, human error, and usability. Table 1 summarizes the operational definitions, indicators, instruments, and scales.

Table 1. Research variables

Variables	Indicator	Instrument	Scale
Efficiency	Provisioning time (seconds) and number of steps	Stopwatch and observation sheet	Ratio
Configuration consistency	Number of deviations from the tier standard	Configuration checklist	Ratio
Human error	Number of configuration errors per process	Observation sheet and system log	Ratio
Usability	System Usability Scale score	10-item SUS questionnaire, Likert 1–5	Interval

A step is defined as a mandatory action that provides input, makes a decision, or advances the process, for example, filling one field, selecting one option, a right-click followed by a menu selection, pressing an execution button, and mandatory tab switching. Logging in, scrolling, cursor placement, closing a popup, and waiting for loading are not counted. This definition holds at the same granularity for both groups.

3.4 Population, Sample, and Testing Protocol

The testing runs in four parts. First, functional testing with the black box method over all features, and the success rate is computed as the percentage of scenarios that match the expected output. Second, efficiency testing by comparing time and number of steps, with an indicator of at least 50%time reduction alongside a reduced number of steps. Third, consistency testing with an indicator of 100% configuration conformity without configuration drift. Fourth, usability testing with SUS involving eight respondents, made up of five regular users and three IT administrators, with an indicator of an SUS score of at least 68. The eight-respondent count follows the property of SUS as reliable on small samples (Bangor et al., 2008; Lewis, 2018). The test design is within-subject because every respondent rates both systems, so each person serves as their own comparator, and statistical power increases without adding respondents.

Efficiency and consistency testing was performed over ten controlled trials per group with locked control variables: an RHEL template already normalized to the bronze tier specification (1 vCPU, 2,048 MB RAM, 40 GB disk), full clone mode, and an IP address via DHCP. The manual group was handled by a full-rights administrator, as the best case for the manual process, while the portal group was handled by a regular user. Provisioning time was split into three segments: t1, user interaction from opening the wizard to submit; t2, waiting for approval; and t3, automatic execution from approval to the active status. The difference test compares t1+t3 on the portal against the manual time as one block, while t2 is recorded separately and reported descriptively in the governance section because its duration depends on when the approver opens the portal, not on system quality.

3.5 Sample-Size Justification

The common rule that demands 60 to 100 respondents applies to surveys that estimate population parameters within a certain margin of error. This study does not estimate a population mean; it evaluates an artifact by comparing two systems rated by the same people. A different purpose demands a different sample-size logic.

For usability testing, eight respondents were set following the property of SUS as an instrument that stays reliable on small samples. Brooke (1996) designed SUS for quick evaluation in industrial settings where respondents are indeed few. Tullis and Stetson (2006) compared several usability questionnaires and found SUS the fastest to reach the same conclusion as a large sample, with high agreement already in the range of eight to twelve respondents. Lewis (2018) confirms that this reliability holds on small samples because the ten SUS items reinforce one another, while Bangor et al. (2008) set its acceptability threshold from an evaluation of hundreds of studies. The within-subject design doubles statistical power because each respondent rates both systems, so between-person variance drops out of the computation.

The same logic holds for the efficiency, configuration consistency, and human error tests, which were run in ten controlled trials per group. These are controlled experiments with locked control variables, not population surveys; therefore, the between-trial variance is small, and ten repetitions suffice to characterize each system. The Mann-Whitney U and Wilcoxon signed-rank tests are designed for small samples and non-normal data (Mann & Whitney, 1947; Wilcoxon, 1945). The adequacy of each sample size ultimately shows in the result, not in a promise before measurement: the efficiency test supports H1 at $p = 1.08 \times 10^{-5}$, and the usability test supports H3 below $p = 0.01$, whereas a sample that is too small would fail to support them. As an honest boundary, this adequacy holds for the main comparison between the portal and Proxmox VE; the split of SUS scores by role, with five non-expert users against three administrators, is reported descriptively because it is too small for an inter-role difference test.

3.6 Data Analysis Technique

The efficiency and consistency data were tested for normality using the Shapiro–Wilk test (Shapiro & Wilk, 1965). Normally distributed data were tested using an independent sample t-test, and non-normal data were tested using the Mann–Whitney U test (Mann & Whitney, 1947). The Mann–Whitney U test was chosen because the portal and manual groups were independent. SUS data were computed using the standard procedure: odd items scored the answer minus one, even items scored five minus the answer, and then the sum of all items was multiplied by 2.5 to yield a range of 0 to 100 (Brooke, 1996). The same respondents rated both systems; therefore, the data were paired. The normality of the score difference was tested using the Shapiro–Wilk test, and normal differences were tested with a paired sample t-test, and non-normal differences with the Wilcoxon signed-rank test (Wilcoxon, 1945).

4. Results and Discussions

4.1 Implementation Result

ExoVirt runs in a production environment serving two data centers, and each data center provides five golden images: four Linux systems and Windows Server 2022. A user submits a request through the wizard, and the system translates the tier choice into a Terraform variables file; Terraform executes a full clone on Proxmox VE, and cloud-init sets the hostname, user, password, and network on first boot. The portal never exposes the Proxmox VE interface to the requester.

4.2 Functional Testing (Black Box)

Black box testing ran 40 scenarios that spanned authentication, user and role management, provisioning and inventory, approval workflow, lifecycle, hardening, and catalog management. Table 2 lists the 40 scenarios and their results. All 40 scenarios produced the expected output; therefore, the functional success rate reached 100%. This figure meets the deployment success indicator of 100%. Comparable web-application research has relied on black-box testing to verify externally observable behavior and API functionality before broader system evaluation (Fattah et al., 2025).

Table 2. Black box test scenarios and results

Code	Test Scenario	Expected Result	Result
A. Authentication and Initial Setup			
A1	Initial installation (create admin)	Admin account created; the wizard self-locks	Passed
A2	Valid login	Signed in and redirected to the dashboard	Passed
A3	Invalid login	Rejected; excessive attempts are throttled	Passed
A4	Logout	Session ends and returns to the login page	Passed
A5	Change password	Password changed; login works with the new one	Passed
B. Request and Approval			
B1	Submit a single VM (valid)	Saved with status Pending Approval	Passed
B2	Submit a VM batch (N units)	N VMs scheduled, named -01 to -0N	Passed
B3	Duplicate VM name	Rejected with a name-taken message	Passed
B4	Boot disk below tier floor	Next is locked with a minimum warning	Passed
B5	Edit and resubmit a request	Changes saved	Passed
B6	Approve a request	Provisioning runs through Terraform	Passed
B7	Reject a request	Status becomes Rejected and is not provisioned	Passed
B8	Revert a request	Request returns to the requester	Passed
B9	Node at critical capacity	Node is disabled or blocked in the wizard and approval	Passed
C. Provisioning and Inventory			
C1	Successful provisioning	VM created in Proxmox and shown Active in Inventory	Passed
C2	View the inventory list	VM list with status is shown	Passed
C3	View VM detail	Configuration, IP, and description are shown	Passed
C4	Reveal credentials	Random password shown and written to the audit log	Passed
C5	Retry a failed VM	Provisioning is repeated	Passed
C6	Sync inventory	VM status refreshed from Proxmox	Passed
D. Lifecycle			
D1	Renew a VM	Expiry date extended and recorded	Passed
D2	Make permanent	VM without an expiry date	Passed
D3	Resize CPU or RAM	VM resources changed in Proxmox	Passed
D4	Edit resources	Changes applied	Passed
D5	Add a disk	New disk appears on the VM	Passed
D6	Harden a VM	Playbook runs and the hardening version is recorded	Passed
D7	Delete a VM	VM destroyed in Proxmox and status Deleted	Passed
D8	Auto-expiry	Enters grace, auto-destroyed, and audited	Passed
E. Discovery and Provider			
E1	Add a provider and test connection	Status Connected	Passed
E2	Run discovery	Nodes, templates, networks, and datastores detected	Passed

Code	Test Scenario	Expected Result	Result
E3	Provider disconnected	Its catalog and network are greyed and removed from the wizard	Passed
E4	Publish resources (CRUD)	Saved and shown in the wizard	Passed
F. Access Management (IAM)			
F1	CRUD user	Changes saved	Passed
F2	CRUD role (RBAC)	Access rights match the role	Passed
F3	CRUD group	Changes saved	Passed
F4	Delete protection	Rejected (409) with a reason	Passed
F5	Environment restriction	Only the allowed resources appear	Passed
G. Audit			
G1	View the audit trail	Activity recorded (who, what, when)	Passed
G2	Filter the audit	Results match the filter	Passed
G3	Export CSV	CSV downloaded with metadata columns	Passed

4.3 Operational Efficiency (Hypothesis 1)

Provisioning one virtual machine through Proxmox VE demands 23 steps, while through ExoVirt, it demands 10 steps, a reduction of 56.52%. Table 3 presents the descriptive statistics of provisioning time for both groups.

Table 3. Descriptive statistics of provisioning time (seconds)

Statistic	Manual (Proxmox VE)	Portal (ExoVirt)
Mean	136.90	98.80
Standard deviation	16.35	6.03
Fastest–slowest value	121–175	up to 110

The Shapiro-Wilk test on the manual time yields $W = 0.837$ with $p = 0.041$, so the data are not normally distributed, and the difference test takes the Mann-Whitney U route. The Mann-Whitney U test yields $U = 0$ with $p = 1.08 \times 10^{-5}$, so H1 is accepted. The two groups do not overlap at all because the slowest value on the portal is still faster than the fastest value in the manual group. This full separation produces a large effect size; therefore, ten controlled trials per group suffice to accept H1 (Mann & Whitney, 1947). The mean time fell by 27.83%. The distribution of both groups is shown in Figure 3.

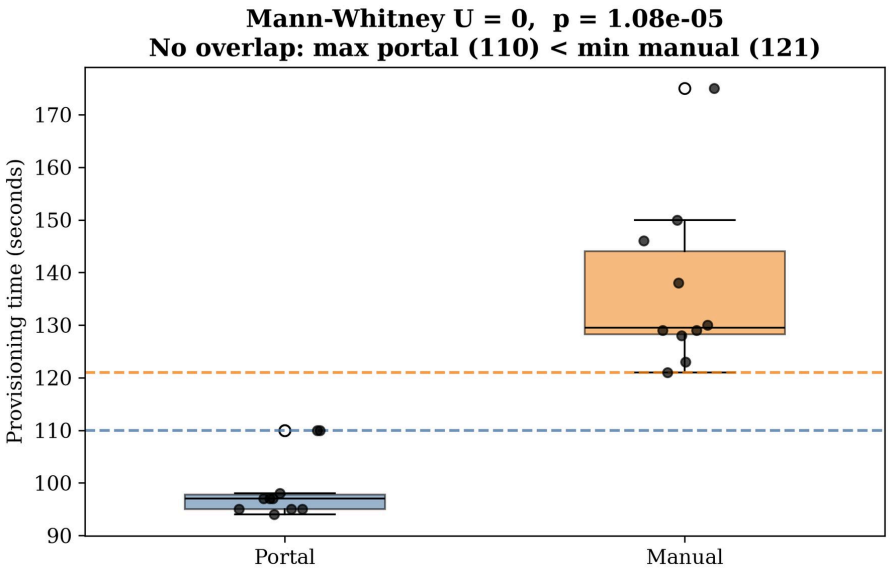


Figure 3. Portal and manual provisioning time with the Mann-Whitney result (U = 0; p = 1.08 × 10^{-5})

Portal and manual provisioning time with the Mann-Whitney result (U = 0; p = 1.08 × 10^{-5}).

The step advantage widens for large requests. The number of steps on the portal does not grow with the number of requested virtual machines because the user only changes one number in the quantity field. A batch trial of ten virtual machines at once built all ten through 10 steps, while the manual approach demanded 230 steps. Completion took 333 seconds on the portal against 1,369 seconds when ten virtual machines are handled one by one manually, and the 1,369 seconds is an extrapolation of 136.90 seconds times ten. The native Proxmox VE interface offers no bulk-request mechanism. The study reports the batch result alongside the per-unit result and does not use it to claim the time-reduction indicator.

The manual group showed a learning effect. The Spearman correlation between trial order and time was −0.857 with p = 0.0015, and the mean of the first five trials, 147.80 seconds, fell to 126.00 seconds over the last five. The manual figure thus represents the best case of the manual process, a full-rights operator who is also the system developer and has practiced ten times.

4.4 Configuration Consistency (Hypothesis 2)

Both groups reached 100% configuration conformity. All 50 parameters across the 10 virtual machines of the manual group matched the specification, and all 50 parameters across the 10 virtual machines of the portal group matched as well, without configuration drift in either group. H_2 is rejected. This null result was predicted before measurement because the control variables demanded a template already set to the Bronze specification; thus, the manual group inherited CPU, RAM, and network with no manual step and left disk as the only parameter open to deviation. This design narrows the room for difference; therefore, the null result holds for this design and cannot be generalized into a claim that the manual process is as consistent as the portal under field conditions that open more parameters to error. An immutable-infrastructure approach that enforces configuration deterministically suppresses configuration drift at a broader scale (Yenikaya, 2026).

4.5 Human Error

The manual and portal groups recorded zero configuration errors across ten trials. The zero in the manual group must be read together with two incidents in the preliminary trials, when the operator miscalculated the boot-disk size increment and when the operator resized the template instead of its clone. Both incidents occurred silently without any warning, and the virtual machine still booted normally; only a manual check revealed them.

The portal advantage on this aspect comes from its interface design. The manual procedure demands that the operator compute the disk-size increment, and Proxmox VE does not validate the result against any organizational standard. ExoVirt removes this field because the boot-disk size is written automatically from the tier definition, so the user never faces a field open to miscalculation. The portal narrows the error surface without removing it entirely, because a wrong tier or template choice remains possible. Its advantage lies in visibility: a wrong tier shows on the review page before submission, while a wrong disk-size increment shows nowhere. This finding aligns with Rahman and Williams (2019), who show that infrastructure code properties determine the emergence of defects, so removing an error-prone field from the interface suppresses the error source.

4.6 Usability (Hypothesis 3)

ExoVirt earned a mean SUS score of 92.19 in the Excellent category, while Proxmox VE earned 37.81 in the Poor category. The Shapiro-Wilk test on the score difference yields $p = 0.1753$; therefore, normality is not rejected, and the difference test takes the paired sample t-test route. Table 4 presents the difference-test results.

Table 4. SUS score difference-test result

Test	Statistic	p	Decision
Paired sample t-test	$t(7) = 6.982$	0.000215	H_3 accepted
Wilcoxon signed-rank	$W = 0$	0.0078	H_3 accepted

All eight respondents gave the portal a higher score, with no single reversal; therefore, H_3 is accepted. The score of 92.19 also exceeds the threshold of 68, and each of the eight respondents individually sits above that threshold. This firm acceptance of H_3 confirms the adequacy of the eight-respondent sample, in line with the literature that shows SUS reliable on small samples (Bangor et al., 2008; Brooke, 1996; Lewis, 2018; Tullis & Stetson, 2006). Splitting the score by role sharpens the finding. Table 5 shows that the rating of Proxmox VE depends on the user's role.

Table 5. SUS scores by respondent role

Role	n	Portal (ExoVirt)	Proxmox VE	Difference
Regular users	5	88.00 (A)	18.50 (F)	69.50
IT administrators	3	99.17 (A)	70.00 (C)	29.17
Inter-role gap		11.17	51.50	

IT administrators gave Proxmox VE a score of 70.00, which exceeds the threshold of 68, while regular users gave the same system 18.50. The 51.50-point difference in rating stems from the difference in user roles. On the portal, the inter-role difference shrinks to 11.17 points, and both roles place it in the Excellent category. The issue this study highlights is the mismatch of Proxmox VE as a self-service interface for non-expert users, whereas as a hypervisor, Proxmox VE remains adequate. The normality of the score difference and each respondent's paired scores are shown in Figure 4.

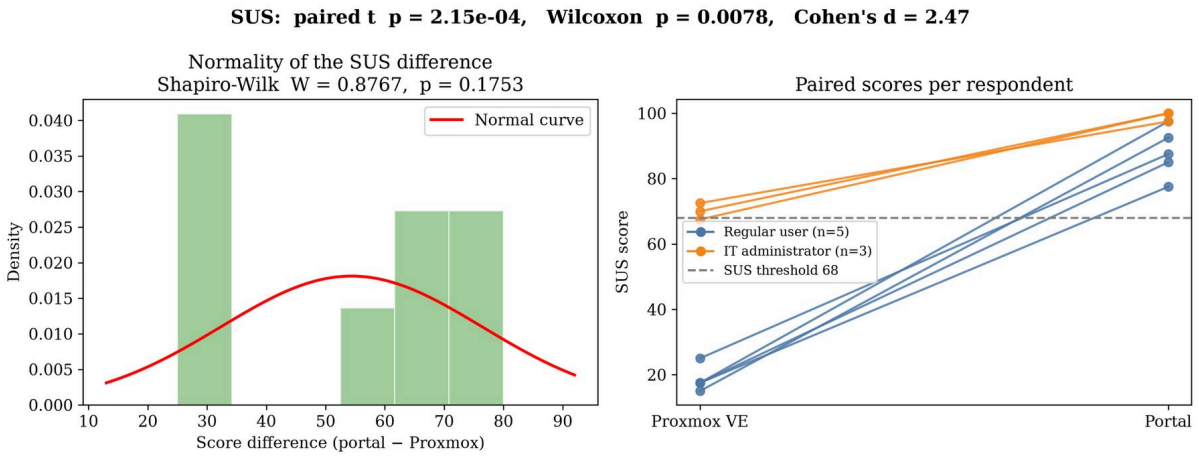


Figure 4. Normality of the SUS score difference and each respondent's paired SUS scores

4.7 Infrastructure Governance

Three roles limit the authority and data scope at the database-query layer; therefore, the restriction applies to the data retrieved, not only to the menu displayed. A Manager sees the virtual machines of the group they manage, and a User sees their own. This model aligns with the RBAC and least-privilege principles (Karatas & Akbulut, 2018; Sandhu et al., 1996). Every regular-user request waits for approval before Terraform runs. Six lifecycle request types, which cover creation, lease extension, resource change, and deletion, place changes to live virtual machines under the same approval gate as new virtual-machine creation. Every event enters an append-only audit trail. In the batch trial, one approval carried ten virtual machines at once along with their audit trail, so governance scales too without adding to the approver's load. Proxmox VE offers only two extreme options: no access at all or full access, and both run without any approval, quota, or logging that links each action to its reason. Activity logging is likewise necessary in role-based web applications because it allows administrators to monitor user actions and strengthen system accountability (Putra et al., 2025). Moreover, system security should be judged not only by technical controls but also by governance capability, documented procedures, and auditable processes (Suroto & Friadi, 2024).

4.8 Indicator Achievement and Discussion

Research Question 1 is answered through layered abstraction that hides Terraform and Ansible, with the number of steps falling from 23 to 10 and the functional success of 40 scenarios reaching 100%. Research Question 2 is answered through unified governance in the form of three roles, approval workflow, six lifecycle request types, and an append-only audit trail. Research Question 3 is answered on efficiency with Mann-Whitney $U = 0$ and $p = 1.08 \times 10^{-5}$, on consistency with an explained null result, and on human error with an architectural error-surface argument. Research Question 4 is answered through an SUS score of 92.19 against 37.81 with the inter-role gap closing from 51.50 to 11.17 points. Table 6 summarizes the achievement of the six success indicators.

Table 6. Success indicator achievement

Indicator	Target	Achievement	Status
Functional success (deployment)	100%	100% (40/40 scenarios)	Met
Reduction in number of steps	Reduced	23 → 10 (−56.52%)	Met
Provisioning time reduction	≥ 50%	27.83%	Missed
Configuration consistency	100% without drift	100% without drift	Met
Configuration error	≈ 0	0	Met
SUS score	≥ 68	92.19	Met

Five of the six indicators were met. The 50% time-reduction indicator was missed at 27.83%, and this study treats its cause as a finding in itself. Approximately 88.7% of the provisioning time on the portal

was machine time, when Terraform runs the full clone and the guest operating system boots. This phase depends on the hypervisor and its hardware specification and is unaffected by the interface that submits the request. ExoVirt uses the same hypervisor as the manual process, so that machine time becomes a lower bound, and no interface can break. The component the portal did cut is human attention time, from 136.90 seconds to 11.20 seconds. Figure 5 separates provisioning time into machine time and human attention time.

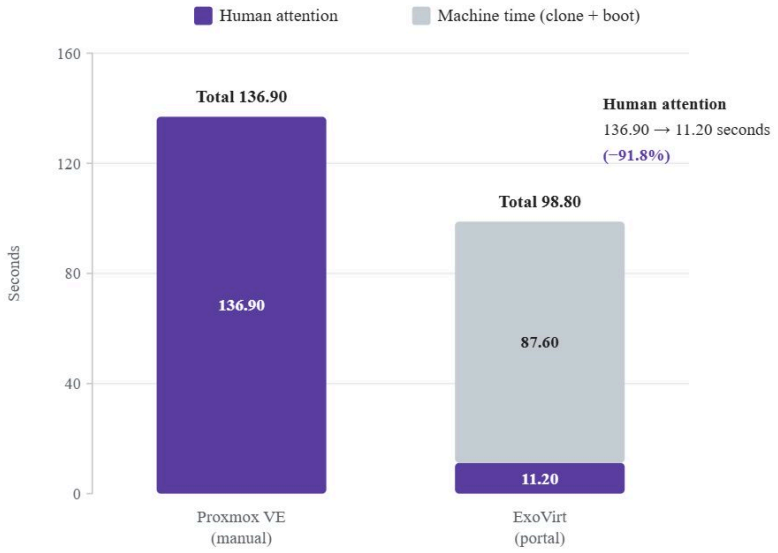


Figure 5. Composition of provisioning time

Hypotheses and indicators must be distinguished. Hypothesis 1 states that there is a significant difference in efficiency without naming a magnitude, and the Mann-Whitney U test accepts it. The 50% time-reduction indicator is a design criterion the researcher set. Hypotheses 1 and 3 are accepted, while Hypothesis 2 is rejected because both groups reached identical, fully conforming configurations under a design that narrowed difference to a single parameter. The miss on one design indicator does not change the decision for Hypotheses 1 and 3. Beyond cutting human attention time, the portal adds guarantees in the form of approval, audit logging, and a narrower error surface that the manual process does not provide.

Efficiency the step count fell from 23 to 10 and remained constant under batch requests, which carried prior automation gains toward the end user. Fadhudin and Ramayanti (2026) cut server-configuration time by 70.4% for one server and 97.6% across fourteen, and Marzuki et al. (2023) reduced human intervention with Ansible, yet both kept the automation inside the administrator's command line. ExoVirt runs the same class of automation but exposes it through a portal, which a non-expert operates; thus, the gain reaches the requester. The machine-time dominance of the portal path agrees with Nabila et al. (2025), who found Terraform bound by provisioning execution time; because ExoVirt runs the same full clone on the same hypervisor as the manual process, that machine time forms a floor no interface can break, and the portal's measurable gain concentrates in the human attention time reported earlier in this section.

Configuration consistency both groups reached 100% conformity; therefore, H2 was rejected under a design that locked an identical Bronze template and left only the disk open to deviation. This null result does not contradict Nelmiawati et al. (2025), who reported that automated hardening yields more consistent configuration than the manual approach, because their comparison opened more parameters than this controlled design. Under field conditions that build from a base image, where every parameter returns to manual entry, the deterministic enforcement of the portal provides

would suppress drift at a broader scale, consistent with immutable-infrastructure evidence (Yenikaya, 2026).

Human error neither group recorded an error across ten trials, yet the portal narrows the error surface by architecture rather than by operator skill. This reading aligns with Rahman and Williams (2019), who show that infrastructure-code properties determine defect emergence: removing the manually computed disk-size field removes an error source that Proxmox VE leaves unvalidated. The advantage lies in visibility, because a wrong tier appears on the review page before submission, whereas a wrong disk increment appears nowhere.

Usability the SUS score of 92.19 against 37.81 places ExoVirt in the Excellent band, and the native interface in the Poor band, one above and one far below the acceptability threshold of 68 (Bangor et al., 2008). The score exceeds the good, acceptable range Cecelya et al. (2024) reported for a user-centered web prototype, and the firm difference on eight respondents matches the small-sample reliability of SUS documented by Lewis (2018) and Tullis and Stetson (2006). The sharper finding is the role gap: Proxmox VE splits its raters by role at 51.50 points, while ExoVirt narrows that gap to 11.17 and lifts both roles into the Excellent band, which locates the problem in the fit of Proxmox VE as a self-service interface for non-experts, not in its adequacy as a hypervisor.

5. Conclusions

5.1 Conclusion

This study built, deployed, and tested ExoVirt as a self-service virtual machine provisioning portal atop Proxmox VE. A self-service interface that hides the complexity of Infrastructure as Code lets non-expert users provision virtual machines themselves, while access control, approval, and audit logging remain intact. The number of steps fell from 23 to 10, the functional success of 40 scenarios reached 100%, provisioning time fell 27.83% with Mann-Whitney $U = 0$ and $p = 1.08 \times 10^{-5}$, configuration consistency reached 100% without configuration drift, configuration error was zero, and the SUS score of 92.19 against 37.81 was confirmed by a paired sample t-test and a Wilcoxon signed-rank test. Hypotheses 1 and 3 are accepted, and Hypothesis 2 is rejected, while five of the six indicators are met. The main finding is the closing of the inter-role gap in the usability rating from 51.50 to 11.17 points, which shows that the portal aligns the non-expert user experience with that of the administrator.

5.2 Research Limitations

The researcher also served as a developer across both measurement groups; therefore, the zero-error figure in the manual group is hard to generalize. The consistency test design narrowed the room for difference to a single parameter; therefore, the null result holds only for this design. The SUS respondent count of eight is adequate for a paired test, but the 5-to-3 composition is too small to test an inter-role difference; therefore, the inter-role finding is descriptive. Comparative data from the VMware environment and from Terraform journals enter only as descriptive comparisons because their hypervisor, operator, and hardware differ.

5.3 Suggestions and Directions for Future Research

Future research should add self-service disk management in the inventory, injection of a user's own SSH key, bulk virtual-machine cloning and import, and power control from the portal. On security, single sign-on integration with Active Directory or LDAP, multi-factor authentication, and automated hardening for Windows would complete the existing scope. On scale, scaling the queue layer with Laravel Horizon through path separation and a per-user concurrency limiter preserves fairness among users, and the Terraform abstraction layer opens multi-hypervisor support. Efficiency and human-error testing should involve operators other than the researcher, usability testing should use more respondents with a balanced role composition, and consistency testing should expand to many parameters beyond disk. This direction aligns with recent evidence from Indonesian information systems, which flags unauthorized access and malware as

prominent risks and prioritizes multi-factor authentication, encryption, staff training, and regular audits.

Acknowledgement

The authors are grateful to Universitas Aisyah Pringsewu and, in particular, the Department of Informatics Engineering, for the academic support behind this study and the system it produced. They also thank the organization that supplied the infrastructure environment and the operational data used in testing; the comparative data from its VMware environment were applied anonymously with the consent of the parties involved. No institution provided dedicated funding for this work.

Author Contributions

AM framed the study, designed and built the artifact, collected and analyzed the data, and wrote the first draft. ZZ supervised the project, validated the methodology, and refined the manuscript. PB checked the results and provided feedback on the draft. AW verified the findings and helped polish the final text. Every author reviewed the completed manuscript and agreed to its submission.

References

Ajmera, K., & Kumar Tewari, T. (2024). Dynamic virtual machine scheduling using residual optimum power-efficiency in the cloud data center. *The Computer Journal*, *67*(3), 1099-1110. https://doi.org/10.1093/comjnl/bxad045

Akbar, M. A., & AlSanad, A. A. (2025). Empirical investigation of key enablers for secure devops practices. *IEEE Access*, *13*, 43698-43715. https://doi.org/10.1109/ACCESS.2025.3549183

Awan, S. M., Azad, M. A., Arshad, J., Waheed, U., & Sharif, T. (2023). A blockchain-inspired attribute-based zero-trust access control model for iot. *Information*, *14*(2), 129. https://doi.org/10.3390/info14020129

Bangor, A., Kortum, P. T., & Miller, J. T. (2008). An empirical evaluation of the system usability scale. *International Journal of Human–Computer Interaction*, *24*(6), 574-594. https://doi.org/10.1080/10447310802205776

Bernardo, J. H., da Costa, D. A., Kulesza, U., & Treude, C. (2023). The impact of a continuous integration service on the delivery time of merged pull requests. *Empirical Software Engineering*, *28*(4), 97. https://doi.org/10.1007/s10664-023-10327-6

Bessghaier, N., Ouni, A., Sayagh, M., Chouchen, M., & Mkaouer, M. W. (2025). Towards understanding code review practices for infrastructure-as-code: An empirical study on openstack projects. *Empirical Software Engineering*, *30*(4), 106. https://doi.org/10.1007/s10664-025-10654-w

Brooke, J. (1996). SUS: A quick and dirty usability scale. *In P. W. Jordan, B. Thomas, B. A. Weerdmeester, & A. L. McClelland (Eds.), Usability evaluation in industry (pp. 189-194). Taylor & Francis*. Retrieved from https://www.researchgate.net/publication/319394819_SUS_--_a_quick_and_dirty_usability_scale

Cecelya, Z., Rahmadi, A. A., & Armin, A. P. (2024). Prototyping antarmuka web cybers academy melalui integrasi desain untuk meningkatkan efektivitas pengguna. *Jurnal Ilmu Siber dan Teknologi Digital*, *3*(1), 43-61. https://doi.org/10.35912/jisted.v3i1.5097

Fadhudin, A., & Ramayanti, D. (2026). Implementasi sistem otomatisasi konfigurasi server berbasis Ansible dan semaphore pada infrastruktur ti perusahaan ritel. *Riau Jurnal Teknik Informatika*, *5*(1), 1-13. https://doi.org/10.30606/rjti.v5i1.4116

Fattah, N. A., Sari, W. E., & Irwansyah, I. (2025). Aplikasi web sistem irigasi dan monitoring lahan reklamasi bekas tambang batubara. *Jurnal Ilmu Siber dan Teknologi Digital*, *3*(2), 181-202. https://doi.org/10.35912/jisted.v3i2.5393

Firmansyah, Y. C., Winarno, W. W., & Pramono, E. (2019). Analisis teknologi virtual mesin proxmox dalam rangka persiapan infrastruktur server. *Jurnal Informa Politeknik Indonusa Surakarta*, *5*(3), 29-35. https://doi.org/10.46808/informa.v5i3.149

Giallorenzo, S., Mauro, J., Poulsen, M. G., & Siroky, F. (2021). Virtualization costs: Benchmarking containers and virtual machines against bare-metal. *SN Computer Science*, *2*(5), 404. https://doi.org/10.1007/s42979-021-00781-8

Hariyadi, I. P., & Marzuki, K. (2020). Implementation of configuration management virtual private server using Ansible. *MATRIK : Jurnal Manajemen, Teknik Informatika dan Rekayasa Komputer*, *19*(2), 347-357. https://doi.org/10.30812/matrik.v19i2.724

Karatas, G., & Akbulut, A. (2018). Survey on access control mechanisms in cloud computing. *Journal of Cyber Security and Mobility*, *7*(3), 1-36. https://doi.org/10.13052/2245-1439.731

Kardha, D., Pamungkas, A. R., & Setiawan, H. (2020). Pengembangan virtual server dengan proxmox ve 6.2 sebagai cloud computing berbasis free/open source software (Virtual Server, Cloud Computing, Proxmox VE 6.2). *Jurnal Ilmiah STMIK AUB*, *26*(1), 85-95. https://doi.org/10.36309/goi.v26i1.126

Khumaidi, A. (2021). Implementation of devops method for automation of server management using Ansible. *Transformatika*, *18*(2), 199-209. https://doi.org/10.26623/transformatika.v18i2.2447

Kumara, I., Garriga, M., Romeu, A. U., Di Nucci, D., Palomba, F., Tamburri, D. A., & van den Heuvel, W.-J. (2021). The do's and don'ts of infrastructure code: A systematic gray literature review. *Information and Software Technology*, *137*, 106593. https://doi.org/10.1016/j.infsof.2021.106593

Lewis, J. R. (2018). The system usability scale: Past, present, and future. *International Journal of Human–Computer Interaction*, *34*(7), 577-590. https://doi.org/10.1080/10447318.2018.1455307

Li, C., Cheng, Y., Zhang, Z., & Yu, P. (2023). Configuration anormaly detection and resolution risk assessment of authoritative domain name server. *Computers & Security*, *135*, 103426. https://doi.org/10.1016/j.cose.2023.103426

Logrippo, L. (2025). Data flow security in role-based access control. *Journal of Information Security and Applications*, *90*, 103997. https://doi.org/10.1016/j.jisa.2025.103997

Mann, H. B., & Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, *18*(1), 50-60. https://doi.org/10.1214/aoms/1177730491

Mao, Y., Fu, W., Zhao, Y., Yuan, Z., Sun, Z., & Zhao, Y. (2025). A zero-trust access control model based on attribute and dynamic trust evaluation for cloud environments. *Symmetry*, *17*(12), 2059. https://doi.org/10.3390/sym17122059

Marzuki, K., Kholid, I. M., Hariyadi, I. P., & Mardedi, A. Z. L. (2023). Automation of open vswitch-based virtual network configuration using ansible on proxmox virtual environment. *International Journal of Electronics and Communications Systems*, *3*(1), 11-20. https://doi.org/10.24042/ijecs.v3i1.16524

Nabila, A., Indrawati, & Davi, M. (2025). Analisis perbandingan Ansible dengan Terraform untuk memanajemen cloud server. *eProceeding of TIK (eProTIK)*. Retrieved from https://e-jurnal.pnl.ac.id/eProTIK/article/view/5771/pdf

Nelmiawati, Gunawan, Arif, H., Haikal, A., & Putra, S. D. (2025). Implementation of server provisioning and hardening automation using Terraform and Ansible with NIST sp 800 – 123. Paper presented at International Conference on Digital, Social, and Science.

Oleksiuk, V. P., & Oleksiuk, O. R. (2021). The practice of developing the academic cloud using the proxmox ve platform. *Educational Technology Quarterly*, *2021*(4), 605-616. https://doi.org/10.55056/etq.36

Orfanou, K., Tselios, N., & Katsanos, C. (2015). Perceived usability evaluation of learning management systems: Empirical evaluation of the system usability scale. *The International Review of Research in Open and Distributed Learning*, *16*(2). https://doi.org/10.19173/irrodl.v16i2.1955

Padur, S. K. R. (2021). From scripts to platforms-as-code: The role of Terraform and Ansible in declarative infrastructure rollouts. *European Journal of Advances in Engineering and Technology*. Retrieved from https://ejaet.com/PDF/8-1/EJAET-8-1-151-156.pdf

Palma, S. D., Nucci, D. D., Palomba, F., & Tamburri, D. A. (2022). Within-project defect prediction of infrastructure-as-code using product and process metrics. *IEEE Transactions on Software Engineering*, *48*(6), 2086-2104. https://doi.org/10.1109/TSE.2021.3051492

Pandey, M. C., & Rawat, P. S. (2024). Virtual machine provisioning within data center host machines using ensemble model in cloud computing environment. *SN Computer Science*, *5*(6), 690. https://doi.org/10.1007/s42979-024-03056-0

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45-77. https://doi.org/10.2753/MIS0742-1222240302

Piki, A., Kasenides, N., & Paspallis, N. (2026). User experience and usability evaluation of an educational mobile application developed for fostering ethics literacy. *Information*, *17*(2), 193. https://doi.org/10.3390/info17020193

Pratama, I. P. A. E., & Putra, P. B. S. W. (2022). Pengujian IaC berbasis devops dan Ansible menggunakan metode black box testing. *Faktor Exacta*, *15*(2). https://doi.org/10.30998/faktorexacta.v15i2.12039

Putra, G. D. A., Rahmadi, A. A., & Armin, A. P. (2025). Rancang bangun sistem informasi manajemen klinik chania care center berbasis web responsif. *Jurnal Ilmu Siber dan Teknologi Digital*, *3*(2), 153-179. https://doi.org/10.35912/jisted.v3i2.5095

Qiu, C., & Shen, H. (2021). Dynamic demand prediction and allocation in cloud service brokerage. *IEEE Transactions on Cloud Computing*, *9*(4), 1439-1452. https://doi.org/10.1109/TCC.2019.2913419

Radhika, E. G., & Sudha Sadasivam, G. (2021). Budget optimized dynamic virtual machine provisioning in hybrid cloud using fuzzy analytic hierarchy process. *Expert Systems with Applications*, *183*, 115398. https://doi.org/10.1016/j.eswa.2021.115398

Rahman, A., Mahdavi-Hezaveh, R., & Williams, L. (2019). A systematic mapping study of infrastructure as code research. *Information and Software Technology*, *108*, 65-77. https://doi.org/10.1016/j.infsof.2018.12.004

Rahman, A., & Williams, L. (2021). Different kind of smells: Security smells in infrastructure as code scripts. *IEEE Security & Privacy*, *19*(3), 33-41. https://doi.org/10.1109/MSEC.2021.3065190

Rajasekar, P., & Palanichamy, Y. (2021). Adaptive resource provisioning and scheduling algorithm for scientific workflows on iaas cloud. *SN Computer Science*, *2*(6), 456. https://doi.org/10.1007/s42979-021-00852-w

Rjeib, H. D., & Kecskemeti, G. (2024). Vmp-er: An efficient virtual machine placement algorithm for energy and resources optimization in cloud data center. *Algorithms*, *17*(7), 295. https://doi.org/10.3390/a17070295

Rzig, D. E., Hassan, F., & Kessentini, M. (2022). An empirical study on ml devops adoption trends, efforts, and benefits analysis. *Information and Software Technology*, *152*, 107037. https://doi.org/10.1016/j.infsof.2022.107037

Samha, A. K. (2024). Strategies for efficient resource management in federated cloud environments supporting infrastructure as a service (iaas). *Journal of Engineering Research*, *12*(2), 101-114. https://doi.org/10.1016/j.jer.2023.10.031

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. *Computer*, *29*(2), 38-47. https://doi.org/10.1109/2.485845

Sandobalín, J., Insfran, E., & Abrahão, S. (2020). On the effectiveness of tools to support infrastructure as code: Model-driven versus code-centric. *IEEE Access*, *8*, 17734-17761. https://doi.org/10.1109/ACCESS.2020.2966597

Shapiro, S. S., & Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, *52*(314), 591-611. https://doi.org/10.2307/2333709

Shokri Habashi, F., Yousefi, S., & Ghalebsaz Jeddi, B. (2021). Resource allocation mechanisms for maximizing provider's revenue in infrastructure as a service (iaas) cloud. *Cluster Computing*, *24*(3), 2407-2423. https://doi.org/10.1007/s10586-021-03262-y

Singh, K. P., Singh, V. K., & Ajmera, K. (2026). Analytical performance evaluation of multi-resource mcdm-based virtual machine scheduling in cloud data center. *The Journal of Supercomputing*, *82*(3), 138. https://doi.org/10.1007/s11227-026-08250-w

Suroto, S., & Friadi, J. (2024). Evaluasi tingkat capability keamanan sistem informasi PT Cppi menggunakan framework COBIT 2019. *Jurnal Ilmu Siber dan Teknologi Digital*, *2*(1), 45-60. https://doi.org/10.35912/jisted.v2i1.2945

Traini, L. (2022). Exploring performance assurance practices and challenges in agile software development: An ethnographic study. *Empirical Software Engineering*, *27*(3), 74. https://doi.org/10.1007/s10664-021-10069-3

Tullis, T., & Stetson, J. (2006). A comparison of questionnaires for assessing website usability. *Proceedings of the Usability Professionals Association (UPA) 2004 Conference*. Retrieved from https://www.researchgate.net/publication/228609327_A_Comparison_of_Questionnaires_for_Assessing_Website_Usability

Venable, J., Pries-Heje, J., & Baskerville, R. (2016). Feds: A framework for evaluation in design science research. *European Journal of Information Systems*, *25*(1), 77-89. https://doi.org/10.1057/ejis.2014.36

Venčkauskas, A., Kukta, D., Grigaliūnas, Š., & Brūzgienė, R. (2023). Enhancing microservices security with token-based access control method. *Sensors*, *23*(6), 3363. https://doi.org/10.3390/s23063363

Wang, J., Wang, Z., Song, J., & Cheng, H. (2023). Attribute and user trust score-based zero trust access control model in iov. *Electronics*, *12*(23), 4825. https://doi.org/10.3390/electronics12234825

Wang, R., Li, C., Zhang, K., & Tu, B. (2025). Zero-trust based dynamic access control for cloud computing. *Cybersecurity*, *8*(1), 12. https://doi.org/10.1186/s42400-024-00320-x

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, *1*(6), 80-83. https://doi.org/10.2307/3001968

Yenikaya, G. (2026). A deterministic, immutable enforcement architecture for ot–it boundaries in critical infrastructure systems. *International Journal of Critical Infrastructure Protection*, *53*, 100854. https://doi.org/10.1016/j.ijcip.2026.100854

Yuan, S., Das, S., Ramesh, R., & Qiao, C. (2023). Availability-aware virtual resource provisioning for infrastructure service agreements in the cloud. *Information Systems Frontiers*, *25*(4), 1495-1512. https://doi.org/10.1007/s10796-022-10302-4

Zhou, N., Georgiou, Y., Pospieszny, M., Zhong, L., Zhou, H., Niethammer, C., Pejak, B., Marko, O., & Hoppe, D. (2021). Container orchestration on hpc systems through kubernetes. *Journal of Cloud Computing*, *10*(1), 16. https://doi.org/10.1186/s13677-021-00231-z