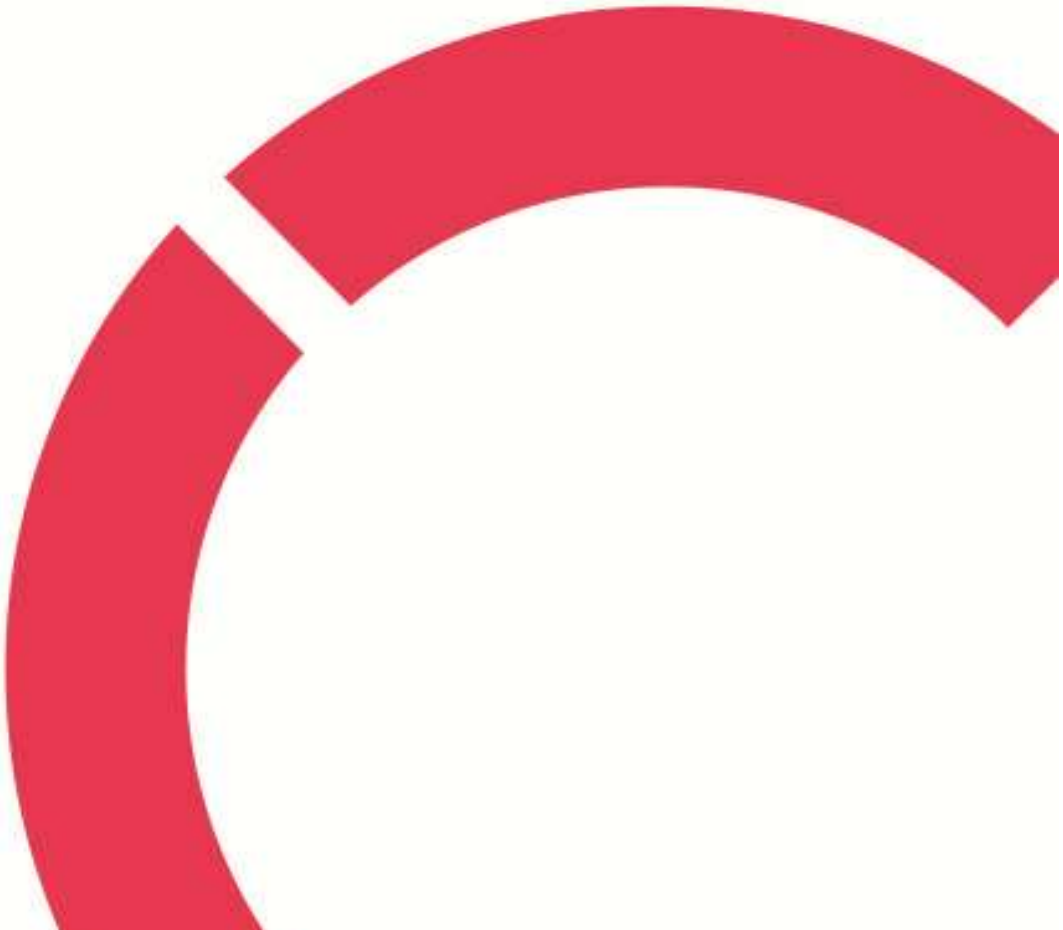
Serhii Didenko

DEPLOYNG MODERN INFRASTRUCTURE FOR SMALL BUSINESSES WITH COOLIFY AND TAILSCALE

Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCES
Bachelor of Engineering, Information Technology
February 2026



ABSTRACT

Centria University of Applied Sciences	Date February 2026	Author Serhii Didenko
Degree programme Bachelor of Engineering, Information Technology		
Name of thesis **DEPLOYNG MODERN INFRASTRUCTURE FOR SMALL BUSINESSES WITH COOLIFY AND TAILSCALE**		
Centria supervisor Maria Kronqvist		**Pages** 33 + 3
Instructor representing commissioning institution or company -		

New open-source applications are useful for small businesses or startups with limited technical resources. Since they need deployment solutions that are easy to use and less expensive, by using Coolify as a deployment platform and Tailscale to create a secure network, businesses will be able to maintain their infrastructure without in-depth knowledge of DevOps. This thesis investigates the implementation of Coolify and Tailscale to facilitate dependable deployment, container orchestration, and secure networking for small-scale enterprises. The goal was to create and test an infrastructure architecture that is easier to use while still being scalable, resilient, and secure. The theoretical framework concentrated on containerization, DevOps methodologies, networking, and open-source deployment platforms, setting up servers, configuring Coolify for hosting applications, and adding Tailscale for secure networking and remote access were all part of the actual implementation. The results demonstrated that small firms could use Coolify and Tailscale together to make it easier to deploy applications, maintain infrastructure automatically, and connect securely from afar without having to spend a lot of money on enterprise-level solutions. The conclusions stressed that open-source solutions are useful and cheap options for digital infrastructure in small businesses.

Key words
cloud-deployment, containerization, coolify, devops, infrastructure, small-businesses, tailscale

CONCEPT DEFINITIONS

API

An application Programming Interface (API) is an intermediary that operates according to a defined set of instructions. Its primary purpose is to enable different programs or services to communicate and interact with each other.

Amazon Web Services

Amazon Web Services (AWS) is a cloud platform that provides a broad range of services that support high range of computing needs. The platform is used for data storage and also for tasks such as renting virtual servers.

MICROSOFT AZURE

Microsoft Azure is a cloud provider positioned as a competitor to Amazon Web Services. It offers similar services but provides stronger integration with other Microsoft products. For organizations that already utilize Microsoft-based infrastructure, Azure is often considered a more suitable option.

CI/CD

CI/CD is a method of autonomous and automated delivery or updating of specific data from point A to point B. this acronym stands for Continuous Integration and Continuous Delivery/Deployment.

CONTAINERIZATION

Containerization is a method of packaging software in an isolated environment that contains all the necessary dependencies.

COOLIFY

An open-source deployment platform that lets people host and manage their own applications on their own servers instead of using proprietary hosting providers.

DOCKER

An open-source platform for building, shipping, and executing apps in containers.

GOOGLE CLOUD

Google cloud is a cloud computing provider. This platform is not very widespread on the market, but it has proven itself well in data analysis, machine learning, and containerization.

INFRASTRUCTURE AS A SERVICE

infrastructure as a service (IaaS) is a strategy for providing cloud services.

OPEN-SOURCE SOFTWARE

This is a model of free code distribution that allows users to freely use and modify it for their own purposes.

PLATFORM AS A SERVICE

Platform as a service (PaaS) is a method of providing cloud services for managing or deploying applications, but the user does not have direct access to the low-level resources.

SOFTWARE AS A SERVICE

Software as a service (SaaS) is a method of distributing cloud computing. A specific application is a service, that is distributed.

VERCEL

A proprietary cloud platform optimized for hosting applications built with the Next.js framework.

VIRTUAL PRIVATE SERVER

This is a virtual environment hosted on real hardware that is provided for use by the client.

VIRTUALIZATION

This technology makes possible to divide the resources of a machine and simultaneously run several different isolated operating systems.

Lock-in

It is a dependence of the technology on the specific infrastructure provided by the technology developer.

ABSTRACT
CONCEPT DEFINITIONS
CONTENTS

1 INTRODUCTION

Cloud computing has become a central concept in modern information technology. It enables convenient access to computing resources and services, with cloud storage playing a significant role in the efficient sharing and management of data across applications and devices. Cloud storage has changed the approach to data storage. Services such as Dropbox, iCloud, and Google Drive presents this transformation. Although cloud technologies provide extensive opportunities for firms, the market is dominated by a small number of corporations. The concentration of control within a few providers presents challenges for small and medium firms. (Yasar, Chai & Bigelow 2025.)

Small and medium companies often face constraints when adopting new cloud technologies due to limited resources. Such constraints hinder their ability to evaluate solutions comprehensively and to anticipate their long-term organizational impact. As noted by Mujalli, Wani, Almgrashi, Khormi and Qahtani (2024), the adoption of cloud systems introduces complex technological and organizational challenges, including compatibility issues, resource shortages, and reliance on external providers. An example of these challenges is the risk of vendor lock-in, where a chosen technology operates only within a specific environment. This dependency restricts flexibility, as applications that rely on proprietary features may not function correctly when migrated to other platforms. Consequently, firms may lose autonomy in decision-making and become constrained by the infrastructure and pricing models of providers. Such limitations underscore the importance of evaluating cloud solutions (Mujalli et al. 2024).

Dependence on technology providers and their infrastructure limits innovation and increases service costs for customers due to monopolistic control over specific technologies, which in turn makes it difficult to modify systems when necessary. Because of these challenges, open-source and self-hosted solutions are becoming increasingly important for businesses that use virtual private servers for their applications, allowing greater control over data, configurations, and infrastructure management. In this way organizations can adapt their infrastructure to operational needs without relying on vendors. The objective of this thesis is to analyse the opportunities and challenges associated with implementing self-hosted solutions as alternatives to commercially managed cloud services. The study aims to identify vendor-independent ecosystems.

2 DESCRIPTION OF ENVIRONMENT

Today, cloud computing has become basis for the infrastructure of most firms. However, an examination of the market for these services reveals that it is largely controlled by three corporations: Amazon Web Services, Microsoft Azure, and Google Cloud. The strength of these providers lies in their ability to offer a comprehensive range of services that meet virtually all organizational needs from renting virtual servers and disk space to complex networking technologies. The dominant position of these providers influences the entire industry, determining the standards and shaping how other companies plan, utilize, and implement cloud solutions in their operations. (Dubendorf 2025.)

Amazon Web Services is the leading provider in the cloud industry, offering the widest range of services and the largest network of data centers. Microsoft Azure is benefiting from strong integration with its enterprise products and hybrid cloud solutions, which makes it attractive to companies already using Microsoft software. Google Cloud, holding a smaller market share, demonstrates strength in data analytics, machine learning, and containerization. (Dubendorfer 2025.) The concentration of power among these providers presents both advantages and disadvantages. While they offer scalability and cost efficiency, their dominant market position raises concerns regarding price increases, vendor lock-in, and the partial or complete loss of control over data. These challenges have led many smaller firms to explore open-source and alternatives that provide greater flexibility.

2.1 Vendor lock-in challenges

Vendor lock-in is still one of the biggest problems that stops more people from using cloud computing. It refers to the dependency that arises when companies use proprietary services. Interfaces, or formats that make it hard to switch to a different provider without spending a lot of time and money on technical issues. Opera-Martins et al. (2014, 92-93) say that cloud providers often make systems that do not work with other systems or cannot be moved to other systems because they use proprietary APIs and closed standards. This approach makes it difficult for clients to move, which means high switching costs and a high reliance on the service.

Vendor lock-in has both technical and strategic implications. Opera-Martins et al. (2014, 95) notes that, from a technological perspective, applications developed on proprietary platforms may require enhanced reengineering to operate on a competitor's infrastructure. From a strategic perspective, organizations risk losing power, as suppliers gain influence when customers become dependent on them. However, infrastructure providers are not the only entities capable of creating vendor dependency. Frameworks such as Next.js demonstrate how developer ecosystems can reinforce these dependencies. Although Next.js is open source, some of its features perform optimally only on Vercel's platform. As a result, users achieve the best performance when tied to a specific vendor, while migrating to a different environment can reduce functionality or increase complexity, thereby discouraging firms from switching providers. These challenges highlight the importance of interoperability and portability. Without such safeguards, small businesses and firms are likely to experience higher costs and autonomy in managing their infrastructure.

2.2 Open-source and self-hosted alternatives

Open-source software has become an important component of infrastructure. Such initiatives make their source code publicly accessible for viewing, modifications, and distributions, distinguishing them from proprietary solutions, where the code is typically restricted and controlled by vendors. This model promotes transparency, openness, and community-driven development (Kalsi 2024). The free code distribution approach is suitable for businesses seeking to avoid dependence on specific infrastructures, software or providers. According to Kalsi (2024), the open-source model offers flexibility, making it easier to implement and maintain. Open-source software is usually supported by a community of developers who contribute to its improvement. This aspect is valuable for small companies, as

it enables them to address their specific challenges independently, rather than relying on provider's solutions. Kalsi (2024.)

Coolify is an example of open-source applications. It is a self-hosted alternative to proprietary software. Coolify can be installed on an organization's own virtual private server, which allows users to manage resources independently and select suitable configurations for their needs. (SkySilk 2024.) Furthermore containerization technologies provide additional flexibility, as containerization enables applications to be placed in isolated environments, making them portable and reducing dependence on specific platforms. This approach facilitates the transfer and deployment of applications across different infrastructures. Hosting such containers on a self-managed servers enables businesses to design customized infrastructures, enhance security and reliability, and maintain full control over resource management. (Susnjara & Smalley.)

All the above makes open-source solutions a very interesting alternative to proprietary solutions. Especially for startups and small businesses for which adaptability and flexibility are very important in today's world, where technologies are developing very rapidly. However, it would be also good to note that open source is not only about advantages; there are also quite a few disadvantages. For example, when problems arise, companies need to find their own solutions, as open-source applications usually do not have very good support because they are not centralized.

2.3 Relevance to small businesses

Choosing the right infrastructure is not simply a technical decision for small firms, but also a strategic one. Companies often cannot maintain complex systems because they lack resources. Buying hardware, setting up cooling systems, securing physical space can be costly. Moreover, hardware failures and downtime represent risks that many firms are not prepared to handle. In this context, virtual private servers offer an alternative by reducing the need for investments, as firms can rent only the resources that are required. Virtual private servers are customizable, providing users with a wide range of resource options, while eliminating concerns related to hardware maintenance, component upgrades, or backup management. (Forward Team 2025.)

Virtual private servers have become relevant due to environmental considerations and continuity requirements. It reduces energy consumption and emissions compared to maintaining multiple physical servers by consolidating workloads into virtualized environments. built-in redundancy and remote data accessibility support firms' models that rely on continuous online operations and remote work. These combined advantages lower costs, greater operational flexibility, reduced dependence on IT staff, and improved resilience make virtual private servers a practical infrastructure solution for firms. Furthermore, this hosting enables enterprises to maintain digital independence by retaining control over their data and deployment environments while addressing technical and financial challenges. (DediRock 2025.) This balance between cost efficiency and autonomy makes it an effective foundation for exploring open-source and self-hosted alternatives discussed in the following chapter.

3 THEORETICAL FRAMEWORK

The theoretical framework explains the key concepts, models that support the study and practical work. Its purpose is to demonstrate how research and the theories relate to the chosen technologies. It defines background such as principles of self-hosting, networking, deployment-automation, and small firms' deployment infrastructure that helps the reader to understand why the selected methods are appropriate. Through this section the practical work is connected to academic knowledge, creating a foundation for analyses in the later chapters.

3.1 Cloud computing fundamentals

When defining cloud computing, the most widely accepted standard comes from the National Institute of Standards and Technology (NIST). NIST characterizes it as a model that allows users to conveniently connect into a collective set of configurable resources, such as servers, applications, or storage, right when they are needed. A core principle is that these resources can be scaled up or down almost instantly by the end users, requiring minimal management or direct contact with the service provider. (Brandao 2018, 87.) This entire model is defined by five key characteristics: it operates on a self-service basis, is accessible over the network, relies on pooled resources for multiple clients, can expand and contract rapidly, and includes a component to measure usage.

Cloud services are usually structured into three main layers. The basic layer is Infrastructure as a Service (IaaS), which provides access to basic resources such as servers and storage. Under the Infrastructure as a Service model, clients are responsible for managing all other components, including operating systems and applications. One level higher is Platform as a Service (PaaS), which offers a procreated environment that enables developers to build and run applications without managing the hardware. At the highest level is Software as a Service (SaaS), which presents fully functional applications, such as Gmail or Salesforce. (Brandao 2018, 87-88.)

Cloud deployment models can be adapted to meet an organization's requirements. A public cloud refers to infrastructure managed by a provider, such as Amaron or Google, and shared simultaneously by multiple clients. In contrast, a private cloud dedicates the entire infrastructure to a single organization, regardless of whether it is managed internally or by an external service provider. also observes that hybrid clouds combine elements of both public and private environments, while community clouds are

designed for multiple firms that share common objectives. cloud computing refers to the delivery of resources over the internet and can be compared to utilities such as electricity or water. This model offers several advantages, including customizability, flexibility, control, and lower costs compared to traditional solutions. However, it also presents certain challenges, regarding security and dependency. (Brandao 2018, 87-88.)

3.2 Virtualization

The main concept of virtualization involves fundamentally changing the traditional operation of computer systems. In a standard configuration, a single physical computer is limited to running one operating system. However, virtualization technology enables a single physical machine to create and operate multiple simulated computing environments in parallel. Each virtual machine runs its own operating system within an isolated environment while sharing the host's physical resources, including the processor, memory, and storage. As a result, one physical computer can perform the tasks of several independent systems, thereby maximizing resource utilization, reducing the need for additional physical hardware, and lowering overall infrastructure costs. (Red Hat 2024.)

The foundation of any virtualization environment is a component known as a hypervisor. A hypervisor, also referred to as a virtual machine monitor, functions as the primary software layer that creates, manages, and operates virtual machines. Its fundamental role is to abstract and allocate the physical machine's resources, such as processing power, memory, and storage, among the virtual machines, while regulating access to them. This mechanism enables multiple operating systems to run simultaneously on a single physical device. The hypervisor ensures complete isolation between these environments, preventing them from interfering with one another. (Susnjara & Smalley 2024.)

3.3 Virtual private servers

Virtual Private Servers utilize virtualization technology to cut a single physical machine into multiple isolated virtual environments. Each virtual private server operates with its own dedicated operating system and allocated resources, offering a level of autonomy and control comparable to a bare-metal server, but at a significantly lower cost. This architecture makes virtual private server a much more effective, secure solution than traditional shared hosting. Furthermore, virtualization increases the efficiency of hardware resource utilization in data centers. Sharing resource of one physical device across

multiple virtual private servers optimizes results in energy savings, maximized hardware lifespan, and reduced operational costs. (Amazon Web Services 2024.)

3.4 Containerization

Containerization is an important technology in cloud computing. It enables applications and all their dependencies to be packaged into small, portable, and independent containers. These containers share the host system's kernel, which simplifies operation and improves performance. This approach differs from traditional virtualization where multiple operating systems run at the same time on separate virtual machines. Containerization ensures consistent application behavior across all environments and development stages and accelerates deployment. (Bell & Castro 2022, 2.)

Docker is recognized as one of the most prominent and influential platforms in the field. Although it was released in 2013, it gained popularity among developers and become an established industry standard. Its success is attributed to its simplicity and portability. Docker provides developers with a powerful mechanism to package any application together with its environment and dependencies into a single container. This container operates consistently across various environments. Such flexibility aligns effectively with DevOps methodologies and Continuous integration and delivery practices, facilitating testing and deployment of new application versions. (Bel & Castro 2022, 3-4.)

3.5 The open-source concept

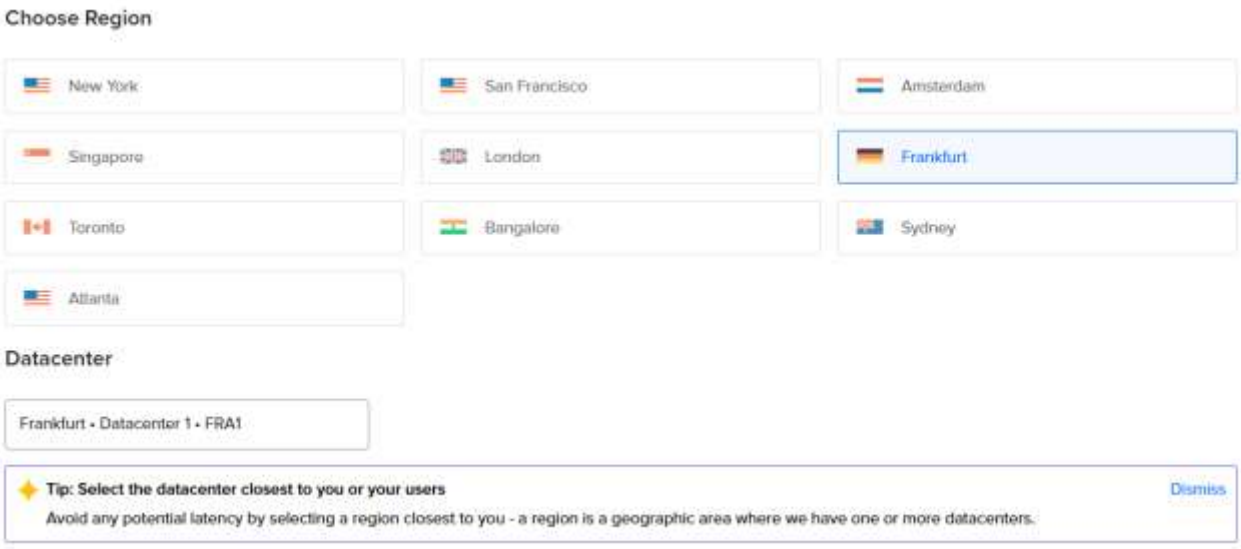
Open source can be outlined in several ways, the most important are transparency and collaboration. This approach contrasts with the proprietary model, in which the company owns its source code as a commercial property. Within the open-source users are granted unrestricted access to software, enabling them to examine the code to understand it. Users are permitted to modify the code, correct errors, and distribute new versions. (Medison 2005, 4.) This mode, unlike the proprietary licensing, brings an environment that encourages innovation. Open-source licenses can be divided into two categories: permissive and copyleft. permissive licenses, such as Massachusetts Institute of Technology (MIT), impose minimal restrictions on developers. Their intent is to allow the code to be freely used. Copyleft licenses, for example the GNU General Public License, work in a different way. They require that any modification of the main source must also be distributed under the same terms. This approach creates a model that preserves the principles of openness and transparency. (Medison 2005, 5-6.)

4 PRACTICAL IMPLEMENTATION

This project shows an approach to creating an independent cloud solution. The objective of this section is to establish a functional infrastructure that enables deployment and management of web applications without reliance on proprietary clouds. The process begins with the configuration of a server, followed by the installation and of Coolify, integration of Tailscale with CoreDNS to secure network and restrict access to the Coolify Dashboard. Finally, the deployment of a WordPress site is presented as a practical example. DigitalOcean, a cloud provider founded in 2012, was used to design the deployment environment. The company focuses on simplifying cloud computing for developers and small and medium firms. According to DigitalOcean (2025), its virtual machines can be created quickly and provide an economical and scalable infrastructure suitable for self-hosted projects. Docker serves as the core of the deployment architecture. Coolify was selected as the self-hosting platform, it is a free and open-source solution that enables the deployment and administration of applications, databases, and services on self-owned or rented servers (Coolify Docs 2025). To secure network communications and enable remote access, Tailscale was implemented. Tailscale is a mesh VPN service. This service allows servers and clients to communicate securely without exposing services to the public internet (Tailscale Docs 2025). These technologies form the foundation for the implementation phase.

4.1 Configuration of the virtual private server

For this project, the server wat initialized in Frankfurt, Germany. This region was chosen because of its convenient geographical location and large data center, which will provide a stable and powerful connection for all users from Europe. The combinations of these factors ensure low latency and stabile connectivity for users across Europe. As can be seen in Picture 1, the Frankfurt data center was selected during the initial deployment stage to ensure optimal performance.

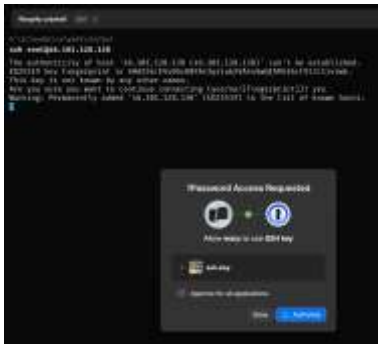


PICTURE 1. Server region selection

When selecting the operating system for the server, Ubuntu 20.04 LTS was chosen. The abbreviation LTS which stands for Long-Term Support is a key factor, as it ensures that the system will receive critical updates and security patches for an extended period, maintaining stability and security without the need for frequent major updates. Ubuntu is considered the most reliable Linux Distributions, known for its stable performance, default security features, and extensive support community. This last aspect was important, as it ensured compatibility with other technologies. For this project it was essential that the operating system function seamlessly with docker and various server management tools, and Ubuntu provides compatibility with most open-source platforms planned for use in the subsequent implementation.

For the server configuration 8GB of memory and 160GB of SSD storage drive ware selected. This amount of resources was determined to be sufficient to ensure the stable operation of the Coolify, the Docker containers it manages, and additional services such as CoreDNS and Tailscale. In terms of cost, this VPS configuration on DigitalOcean amounts to approximately 48$ per month. It is worth noting that, although DigitalOcean is recognized for its intuitive interface and reliable performance, it is not the most cost-effective option on the market; competitors such as Hetzner offer comparable functionality at a lower price. However, the choice of this platform for this study was pragmatic, as student credits provided by the platform made its use cost-free and therefore suitable for the purposes of this research.

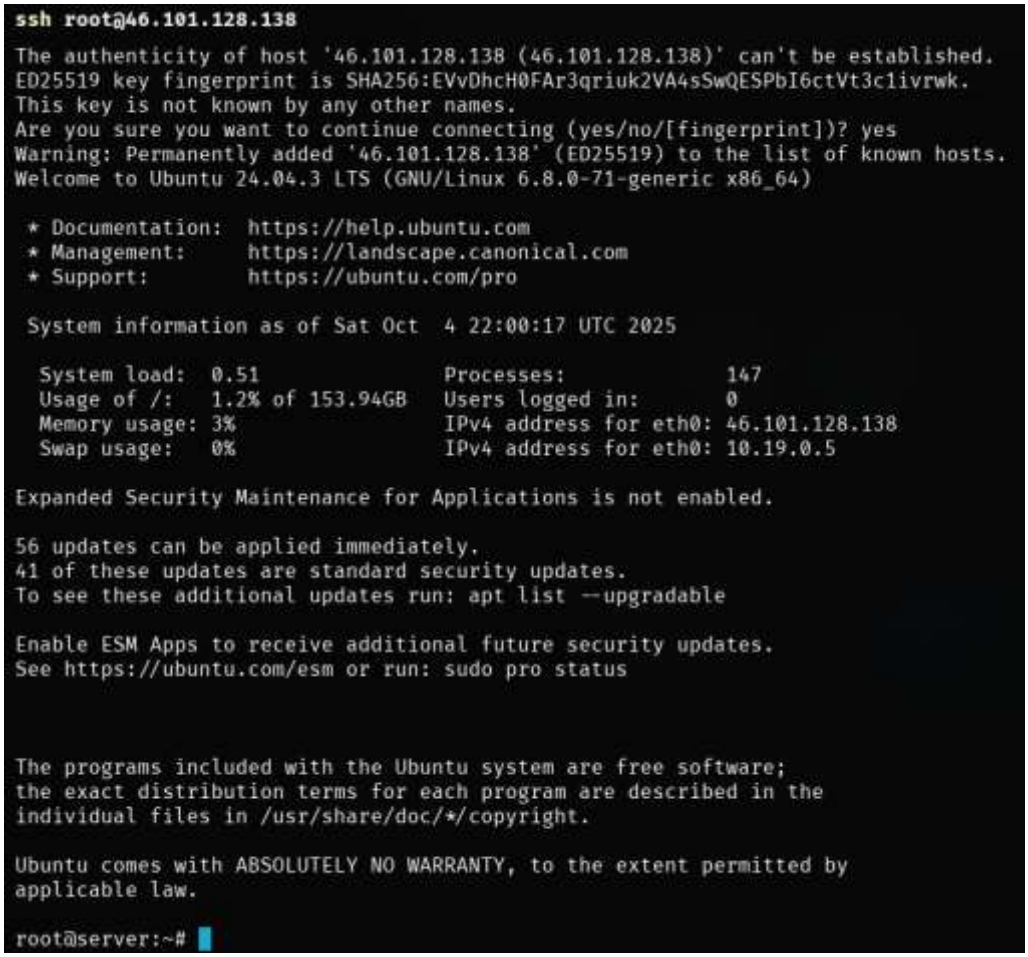
For authentication SSH key were chosen instead of password-based login, SSH keys provide stronger protection against unauthorized access. SSH id a cryptographic network protocol used to establish secure connections between devices over untrusted networks (Cloudflare Docs 2025). The SSH key was generated using the built-in functionality of password manager application, which simplifies key management while ensuring that the private key remains securely stored. Depending on the operating system, SSH key can also be created using command-line utilities or tools such as PuTTYgen. The public key was then added to the DigitalOcean account, allowing secure remote access to the VPSA without the use of passwords. The process of establishing a connection to the virtual private server using an SSH key is illustrated in Picture 2.



PICTURE 2. Connection to the server via ssh

After completing all the initial configurations of the virtual private server and successfully verifying the ability to connect, its deployment process was considered successfully completed. On the prepared server, the installation and configuration of the Coolify ware initiated, followed by the necessary domain configuration procedures and the enhancement of network security through the implementation

of Tailscale. As shown in Picture 3, a successful remote connection to the server confirms that the secure SSH connection is verified and the server deployment is complete.



PICTURE 3. successful remote connection to the server

4.2 Installation and setup of Coolify

The next step in the implementation process was to install Coolify. Which was performed after successfully connecting to the server. Docker serves as the platform on which Coolify operates, providing a graphical interface for managing containerized environments without requiring command-line configuration. Installation process was initiated by executing the command in the terminal. The installation script used to deploy Coolify on the server is presented in Code 1.

```
curl -fsSL https://cdn.coollabs.io/coolify/install.sh | sudo bash
```

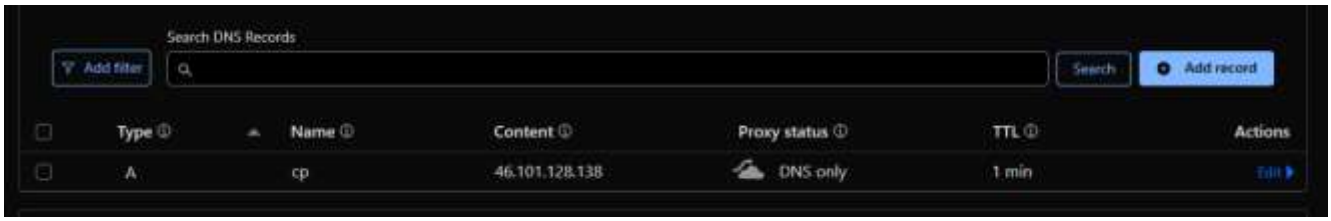
Code 1. Coolify install script

This script automatically downloads and installs all the components required for Coolify to operate. The installation procedure is fully automated, which reduces the likelihood of configuration errors and ensures that the environment is set up according to the latest stable software version. Upon completion of the installation, the terminal displays the public IP address of the server hosting the Coolify instance. This address can be entered into a web browser to access the Coolify interface. During the initial login, the user is guided through a setup process that includes creating an administrator account and defining a password. The successful completion of the installation and the output displayed in the terminal are illustrated in Picture 4.

PICTURE 4. the installation of coolify is complete

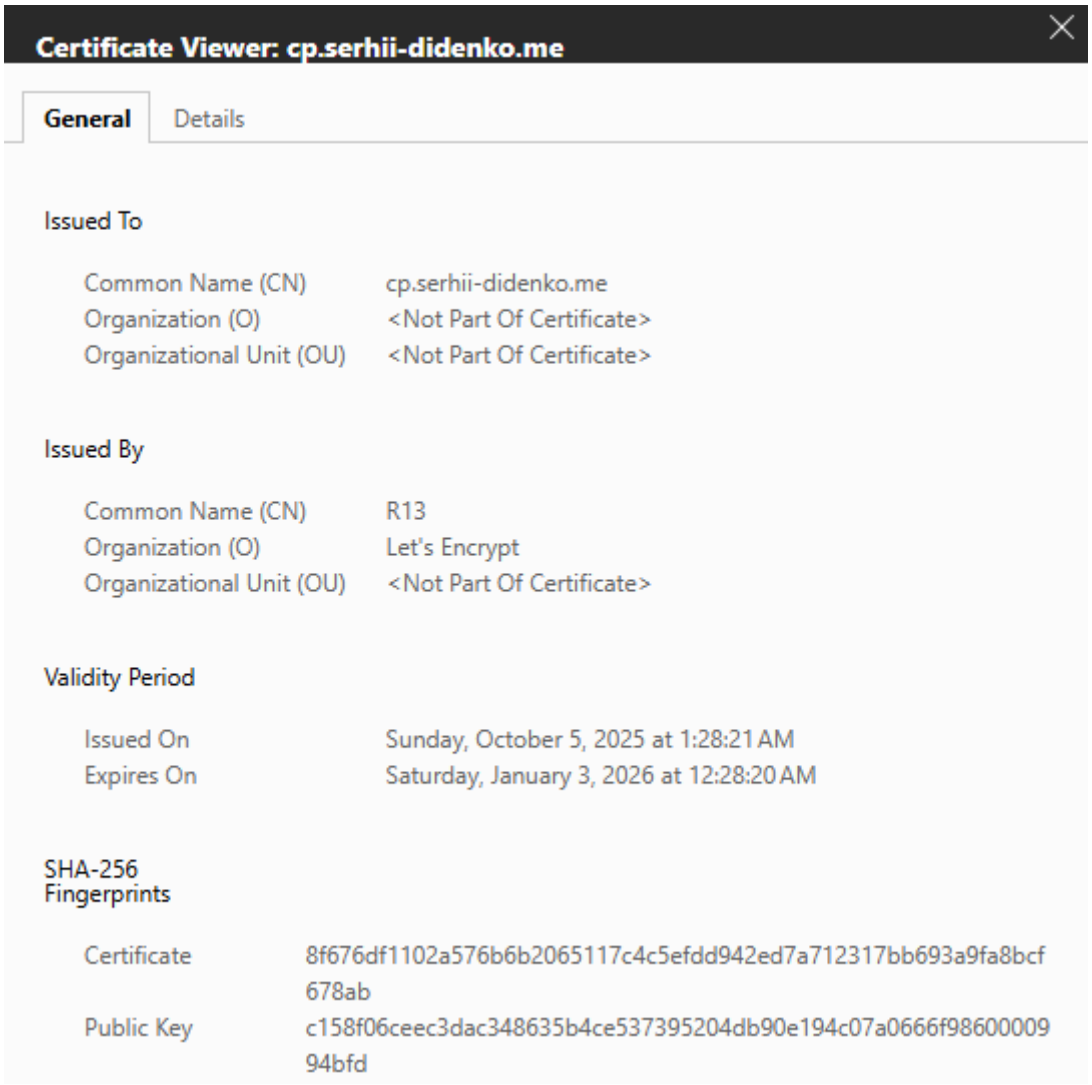
4.3 Domain and SSL integration

After Coolify was installed and configured, the next step is to add a domain to replace the default IP address and enable HTTPS access. The Coolify instance was made accessible through the subdomain cp.serhii-didenko.me, which improved usability. The configuration process began by linking the domain to the IP address of the server. The Cloudflare DNS control panel was used as the nameserver for this purpose. Within Cloudflare, an A record was created to associate the subdomain with the server's IP address, allowing external users to access the server by domain instead of its numeric address. Once the settings were saved the updated DNS propagated across the network, completing the connection between the domain and the Coolify instance. The setup of the DNS record in the Cloudflare control panel is illustrated in Picture 5.



PICTURE 5. Cloudflare DNS configuration

The domain was added within the Coolify interface after DNS propagation had been completed. In the Coolify interface: by navigating to settings > general, and inserting the full URL to the domain field, was entered as https://cp.serhii-didenko.me , the configuration was finished by clicking the save button. After that step Coolify automatically installed the SSL certificate using Let's Encrypt. The certificate creation process is fully automated and takes only a few minutes to complete. The SSL certificate ensures that all communication between the user's browser and the Coolify is encrypted.



PICTURE 6. SSL certificate status

The Coolify become accessible through the domain, once the SSL certificate had been installed. After the domain was linked to the Coolify, the server could also be accessed using its domain instead of IP. Picture 6. SSL certificate status demonstrates the installation and activation of the SSL certificate within the Coolify. Using the domain rather than the IP address simplifies administration and improves readability, provides a consistent access method even if the IP changes. The command used to connect to the virtual private server via SSH using the domain instead of the IP address is shown in Code 2.

ssh root@cp.serhii-didenko.me

Code 2. SSH login to the server

4.4 Setup of Tailscale and CoreDNS

The secure network for Coolify was created using Tailscale and CoreDNS, which together provided an encrypted private network and a reliable internal name-resolution system. The objective of this stage was to ensure that administrative access to Coolify and supporting containers was available only through the private network, while public services remained isolated from the management interface. The process began with the installation of Tailscale on both the client and server operating systems. On the server running Ubuntu, Tailscale was installed using the script shown in Code 3. After the installation, the command sudo tailscale up was executed. After completion, the terminal displayed a unique authentication link. By opening this link in a web browser and signing in with a Tailscale account, the server was successfully added to the private network. The connection was then verified using the commands tailscale status and tailscale ip -4, which confirmed the private address.

```
curl -fsSL https://tailscale.com/install.sh | sh
```

Code 3. Tailscale install script

The client was configured using the software installer downloaded from the Tailscale site. After logging in with the same account, the client's device appeared in the Tailscale's dashboard. This allowed the administrator to connect to the server using a private IP, this method will increase security. To enhance protection, the command sudo tailscale up –ssh was executed to enable Tailscale's built-in SSH service, which allows devices in the private network to create SSH connections without relying on the public keys or exposed network ports. The command sudo ufw deny 22/tcp was used to close the standard SSH port through the firewall, ensuring that all future administrative access occurred only via the Tailscale network. After these configurations the server become invisible to external SSH scans, and the command ssh root@100.88.58.77 confirmed that secure login was functioning through Tailscale's network. The connection process between the client and the server is demonstrated in Picture 7, which confirms that the secure login is functioning through the Tailscale network.

```
ssh root@100.88.58.77
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.8.0-71-generic x86_64)
```

PICTURE 7. Ssh through the Tailscale VPN

After network configuration the next step was to configure how Coolify manages its internal communication. coolify operates within a dedicated docker bridge network used to control its containers. When the command docker network inspect coolify was executed, it showed the network topology and the subnets assigned to internal components. Understanding this structure was important for configuring split-DNS resolution, which maps private service names to their respective bridge addresses, while public domains continue to resolve the external ip. To implement this, CoreDNS was installed as a lightweight DNS resolver, configured to bind only to the tailscale interface. The configuration files were placed in the directory /opt/coredns and defined within a file named corefile. The configuration used in the environment is presented in Code 4. the newly created DNS resolver was then added as a custom DNS server in the Tailscale panel to enable internal name resolution through the private network.

```
serhii-didenko.me {
  bind 100.88.58.77
  log
  errors
  hosts {
   100.88.58.77  cp.serhii-didenko.me # coolify on ip v4
   fd7a:115c:a1e0::4f01:3a50 cp.serhii-didenko.me # coolify on ip v6
   10.0.1.6  coolify.serhii-didenko.me # an example of a service
   10.0.1.5  db.serhii-didenko.me # an example of a service
   10.0.1.3  redis.serhii-didenko.me # an example of a service
   10.0.0.2  sentinel.serhii-didenko.me # an example of a service
   fallthrough
  }
  forward . 1.1.1.1 8.8.8.8
  cache 120
}
. {
  bind 100.88.58.77
  forward . 1.1.1.1 8.8.8.8
  log
  errors
}
```

Code 4. coreDNS config file

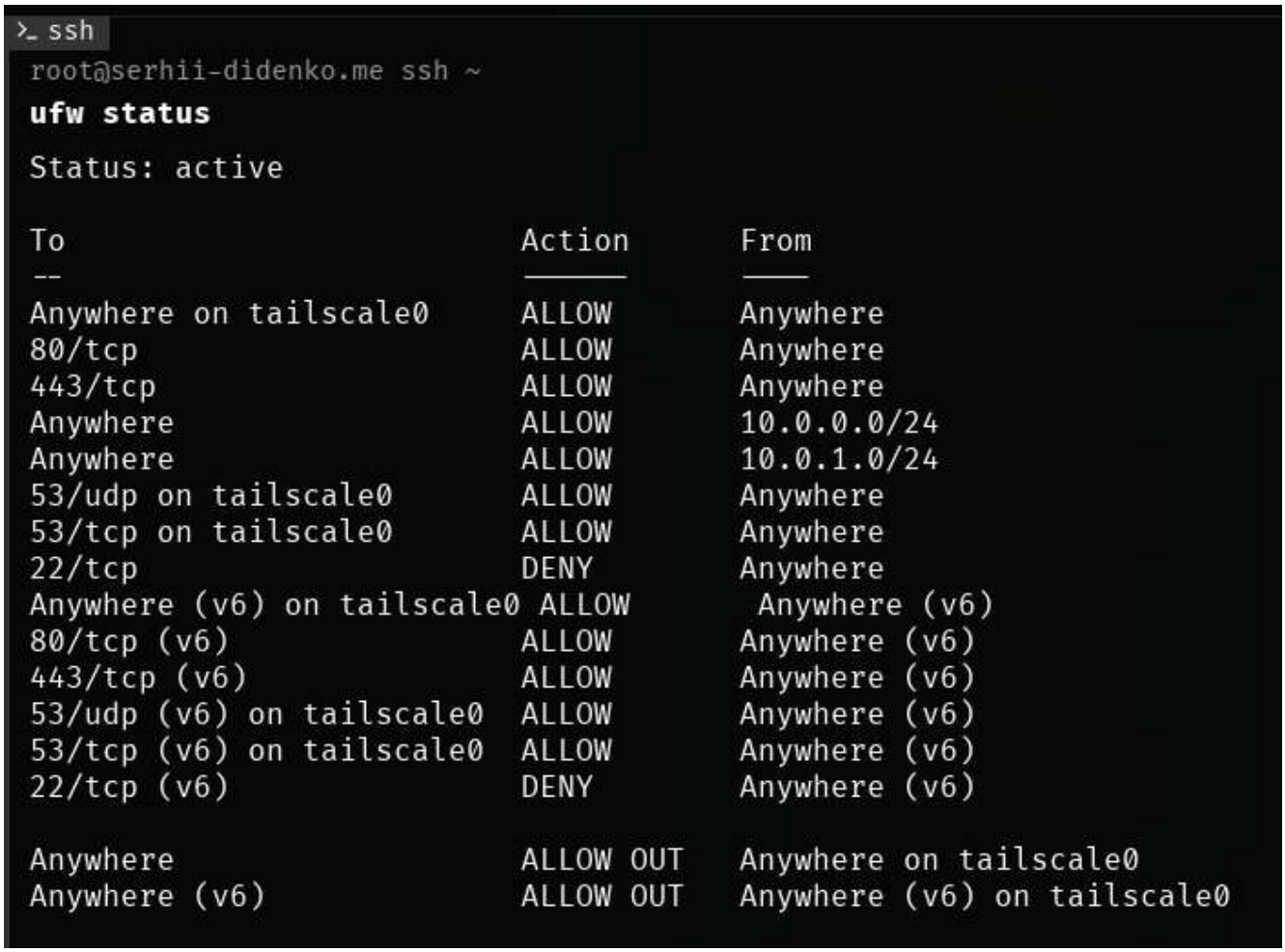
The configuration defines the domain serhii-didenko.ne as an internal DNS zone and restricts DNS binding to the private Tailscale address. Each internal service is assigned a hostname corresponding to

the private network address or to a Docker bridge address. External queries are forwarded to Cloudflare and Google's public DNS. While a short cache interval improves performance for repeated lookups. A second zone, represented by the dot, manages general DNS forwarding, allowing the host system to utilize CoreDNS for non-local queries. To ensure persistent operation and simplify maintenance, the CoreNDS service was deployed as a Docker container. The configuration used for this deployment is presented in Code 5, which shows the Docker Compose file for the CoreDNS service.

```
version: "3.9"
services:
  coredns:
    image: coredns/coredns:latest
    container_name: coredns
    restart: always
    network_mode: host
    volumes:
      - ./Corefile:/Corefile:ro
    command: -conf /Corefile
```

Code 5. Docker-compose config file for the coreDNS service

After running the container with docker compose up -d, the service becomes active and begins responding on port 53 over the Tailscale interface. The command dig +short cp.serhii-didenko.me@100.88.58.77 returned the expected IP, confirming that DNS resolution was correct. To maintain network isolation, the host firewall was configured using UFW. The firewall remained active and blocked public access to ports 80 and 443, allowing only DNS and administrative connections through the Tailscale interface. The firewall configuration applied on the server is illustrated in Picture 8.



PICTURE 8. UFW status

The configuration restricted all DNS activity to the Tailscale interface, disabled external SSH access, and retained only the necessary web ports for public content delivery. The public IP handled solely web requests, while all administrated and internal communication remained confined within the private network. The functionality of the configured system was verified following deployment. Executing the command dig +short cp.serhii-didenko.me returned the private address, while other internal services, such as db. Serhii-didenko.me, resolved to their docker bridge IP's. The verification process, demonstrating the correct internal and external address resolution, is shown in Picture 9. Accessing cp.serhii-didenko.me in a browser redirected to the Coolify, confirming that the Coolify was accessible only from devices within the private network. The SSH connection ssh root@cp.serhii-didenko.me also succeeded, demonstrating that remote management could be performed over the encrypted private network.

PICTURE 9. Cli command dig for the private and public ip demonstration

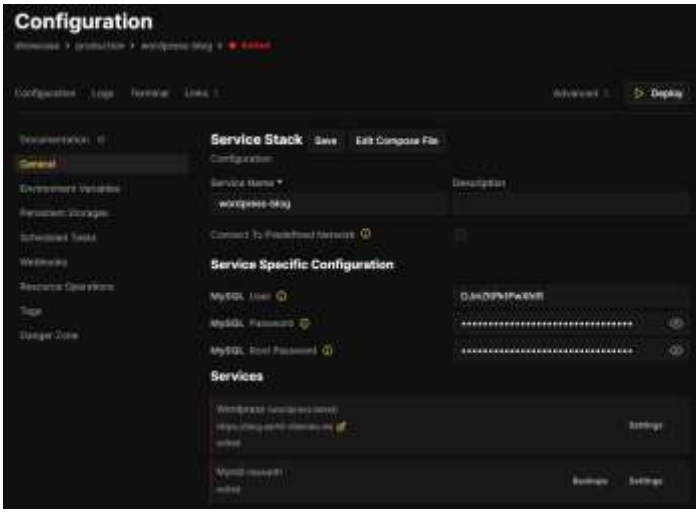
When new applications were deployed, additional entries were added to the CoreDNS host's section. After this step CoreDNS service was restarted by executing command docker compose restart to apply changes. This setup gives possibilities to integrate new sites while keeping the security of the network. This configuration divides public and private areas. Coolify operated in the private network, utilizing internal IP addresses and DNS managed by CoreDNS. Ports 80 and 443 remained open to host public websites on the server. This architecture ensured that the administrative interface was not publicly exposed, that all remote connections ware encrypted, and that maintenance was simplified through the use of domains.

4.5 Deployment of applications

The main function of the infrastructure is to host applications. After the installation of Coolify and the configurations, the environment were prepared to host applications. A WordPress site was deployed using Coolify's template system to demonstrate the deployment process. This example illustrates how applications can be launched with minimal configuration. The deployment began with the creation of a subdomain blog.serhii-didenko.me. to add the subdomain in the Cloudflare, an A record was created pointing to the same public IP address as the Coolify, ensuring that users could access the WordPress through a domain. The DNS record setup is shown in Picture 10, which illustrates the Cloudflare DNS configuration for the blog subdomain.

PICTURE 10. Cloudflare DNS configuration for the blog subdomain

After the new domain was configured in the Coolify, on the projects page, new project was created. Coolify provides a wide range of preconfigured templates for applications. The WordPress template was selected to automate the setup of both the application and its corresponding MySQL database. Coolify then deployed a containerized environment consisting of two services. The process of the basic configuration of the WordPress deployment template is illustrated in Picture 11.



PICTURE 11. Deployment of WordPress through Coolify template selection

After the selection of the template, several configuration parameters were added. The application was named wordpress-blog, and the domain was set to blog.serhii-didenko.me. the container's environment variables included database credentials, the site title and administrator account details. Once the deploy button was clicked, the build process began. Coolify automatically retrieved the required images, created the necessary containers, and configured internal network, when the deployment was completed, the status of both the Wordpress and MySQL containers changed to running. The applications become publicly available via the domain. To maintain internal connection, the new domain was also added to the CoreDNS configuration, mapping blog.serhii-didenko.me to the private IP. The configuration part demonstrated in Code 6. Config for new subdomain in the CoreDNS service

```
hosts {
   100.88.58.77          blog.serhii-didenko.me
   fallthrough
}
```

Code 6. Config for new subdomain in the coreDNS service

After saving all the configuration updates, the CorDNS service was restarted by the command docker compose restart. To verify the new configuration's update, the command dig +short blog.serhii-didenko.me @100.88.58.77 was completed, it returned the private IP. This proved that the Domain Name System name resolution system was functioning within the private network. The verification process, demonstrating the correct internal DNS response for the configured domain name, is shown in Picture 12.

PICTURE 12. DNS resolution test for blog.serhii-didenko.me through CoreDNS

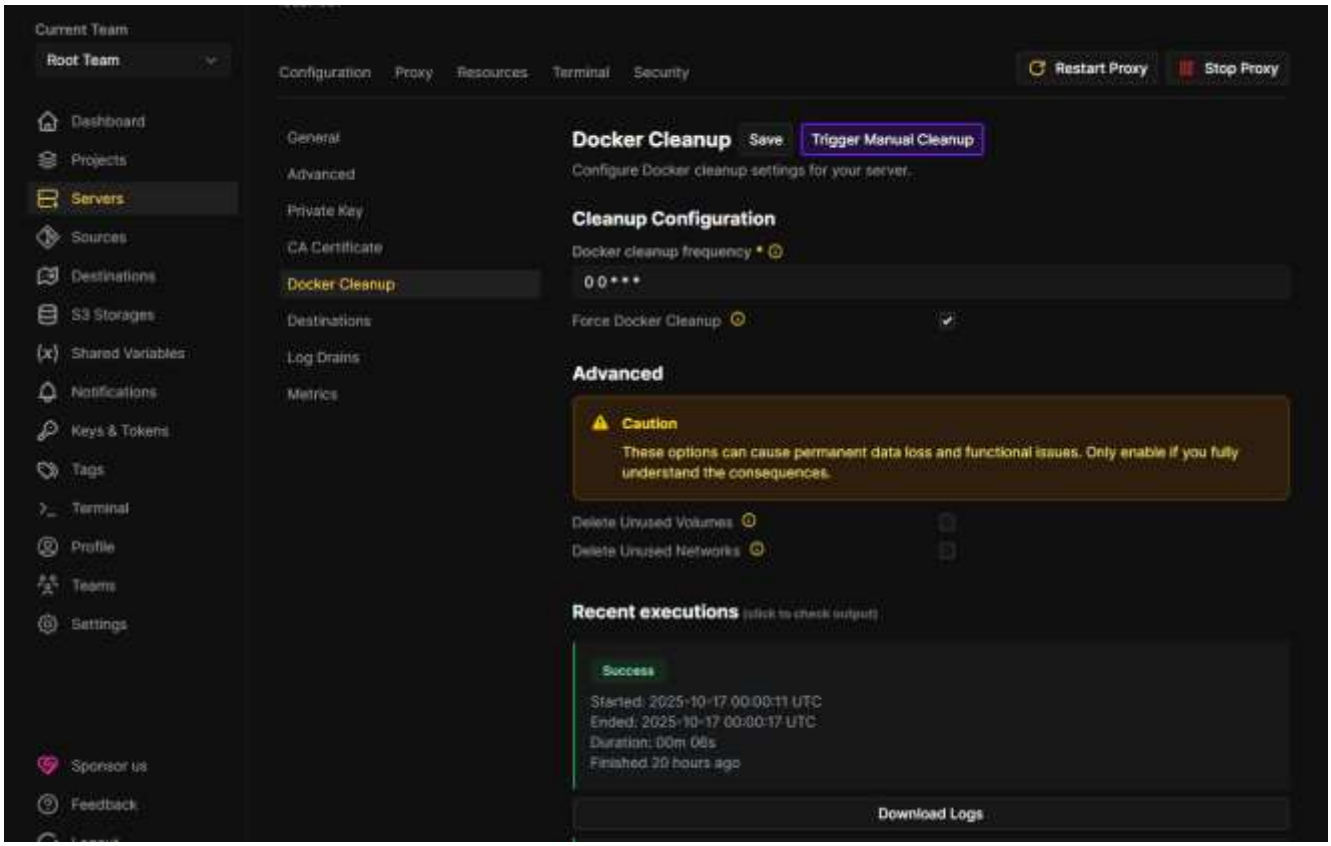
The deployment process demonstrated the capabilities of the Coolify solution, which managed a complex application. As a result, Wordpress was installed and configured, with Coolify automatically handling all tasks. Overall, this combination of tools demonstrated an important conclusion in practice: small business can host complex applications on their own infrastructure. This enables them to avoid reliance on large cloud providers and maintain full control over their data and systems, while still meeting standards for applications stability.

4.6 Security and maintenance

It is important to understand that deploying infrastructure is only the first step. To ensure long-term stability, it cannot be just configured, it requires continuous attention through regular monitoring and maintenance. Coolify simplify the application deployment, but it does not eliminate human responsibility. Administration process still played a main role and is accountable for several tasks: storing all passwords, access keys, update containers, server monitoring. This section demonstrates the key measures implemented to maintain system. Security was enhanced through the configuration of two factor authentication for the login protection. the system was maintained by regularly executing Docker cleanup function to remove unnecessary containers. Server performance was monitored using the metrics function.

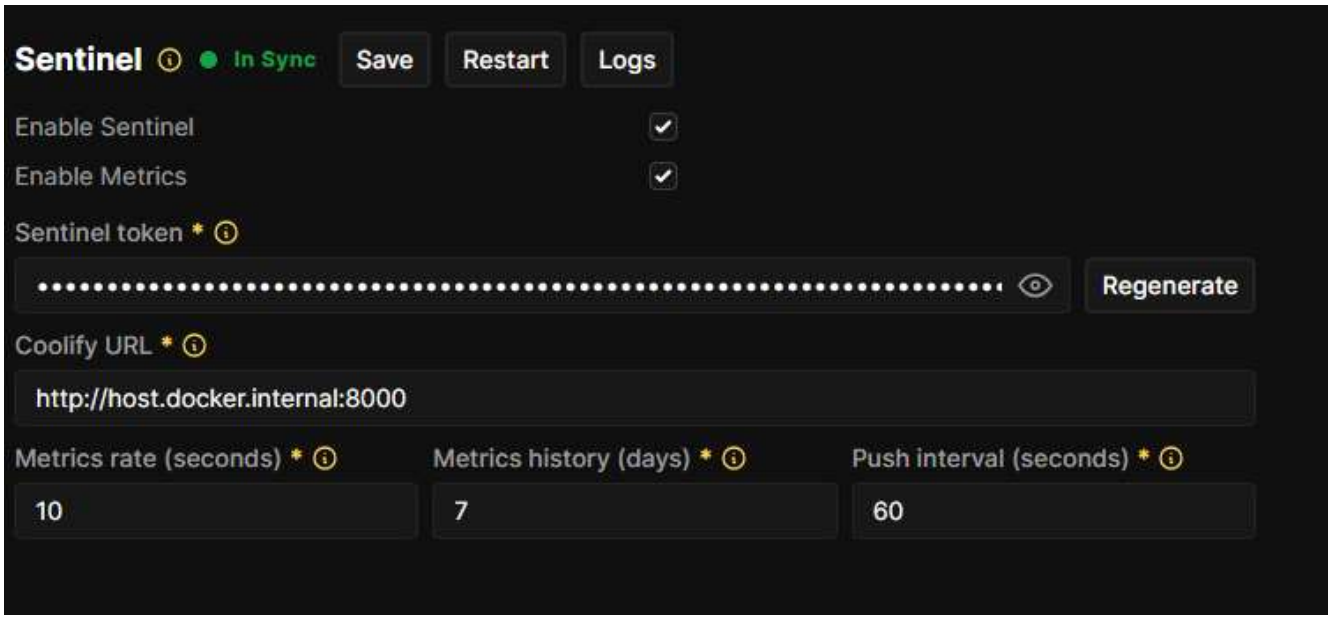
To enhance the security, two factor authentication was turned on. The setup process was within the profile settings, the configuration was started, and Coolify showed the QR code, an authenticator application on a phone was used to scan the QR. The generated code was entered to the Coolify to complete the configuration process. After that step all logins required login and password and one-time numeric password from the passsword manager on a phone. This step reduced the risk of unauthorized access to the dashboard, even in cases where the password might be compromised.

Maintenance of the container environment is important, because Docker accumulates unused containers and volumes that consume storage. Coolify includes feature that allows this cleanup operation without using the command line. This feature can be accessed on the server’s general page, function called Docker cleanup. By clicking the button, the process freed up space and ensured that only active containers and current images remained in use, helping the system maintain optional performance. The cleanup interface and the execution of the maintenance process within the Coolify admin panel are presented in Picture 13.



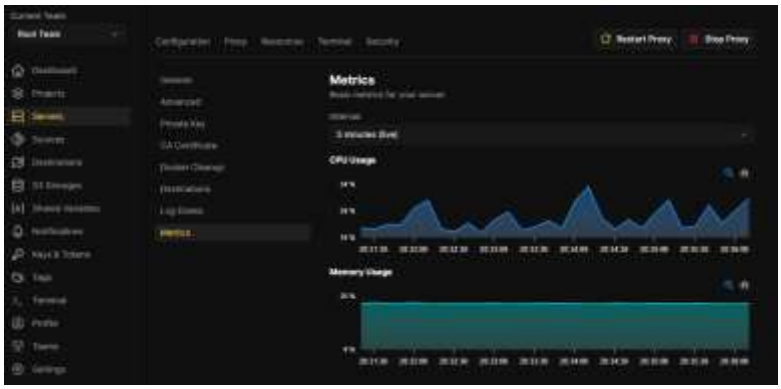
PICTURE 13. Manual Docker cleanup operation in Coolify

Coolify includes a Metrics module that provides continuous monitoring of resource usage and application performance. This feature displays system statistics such as CPU load, memory usage and container activity. To enable this feature, metrics must be turned on, in the settings page. The configuration is in the general settings, Sentinel section. After the enabling the function and saving the settings, a new metrics tab appears in the server page, allowing real-time observation of system performance. The activation process within the Sentinel configuration is presented in Picture 14.



PICTURE 14. Enabling Metrics collection in Sentinel settings

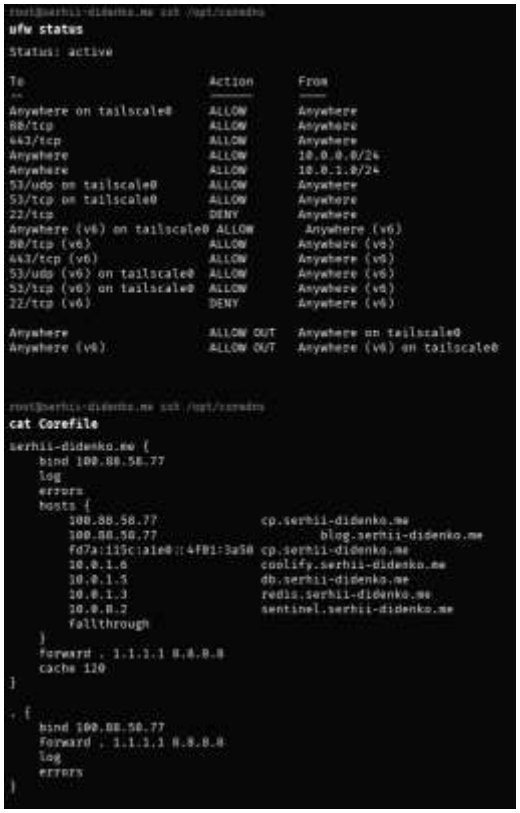
The metrics displays charts of general server performance. This function improves system visibility but also requires careful consideration. Data collection and visualization increase resource consumption, on systems with limited memory or processing capacity. For this reason, enabling metrics is recommended in production environments where the advantages of monitoring outweigh the additional computational load. In smaller self-hosted installations, periodic manual inspections may be sufficient to maintain an appropriate balance between visibility and resource efficiency. The monitoring interface and server resource utilization are illustrated in Picture 15.



PICTURE 15. Metrics dashboard displaying server resource utilization

4.7 Evaluation of the environment

The evaluation of the deployed environment focused on confirmation that the infrastructure created using Coolify and Tailscale was functional, with the goal of ensuring that the solution met the objectives of simplifying application deployment. The testing began by examining the stability of Coolify during deployment: the Wordpress site was correctly configured and continued operating without user intervention. Coolify's management system automatically handled container resources. This demonstrated that Coolify's orchestration layer isolates applications, preventing failures in one service from affecting others. The Tailscale network was also evaluated for security, all administrative connections were made only over the private network. Configured firewall rules blocked all external access attempt on port 22, while the CoreDNS resolver returned internal IPs for the private network requests and resolved public DNS for external domains. This ensured privacy and flexibility, administrative access remained private, while public content stayed available to users. The dual-resolution mechanism is demonstrated in Picture 16, which shows the verification of private and public DNS resolution via CoreDNS.



PICTURE 16. Verification of private and public DNS resolution via CoreDNS

The functionality of the configured two factor authentication was tested, and it performed as a reliable security mechanism. When login attempts were made using incorrect or expired verification codes, Coolify detected the issue and denied access. This confirmed that this system had been implemented correctly and that all users' sessions were protected. HTTPS encryption was already enabled through certificates ensuring that all data transmitted between users and applications remained confidential and protected from unauthorized access. To evaluate the system's performance the metrics was used to monitor server resources. After activating monitoring function, CPU and memory consumption increased but remained within acceptable limits. No issues were encountered while performing standard maintenance tasks, such as cleaning Docker of unnecessary images or restarting services. This evaluation led to a conclusion that using the Coolify for regular maintenance is safe. This method is not only convenient but also acts as an additional safeguard, helping to prevent errors and unnecessary usage of server resources.

4.8 summary of the practical work

The practical work demonstrated how, by using an open-source tools, it is possible to build infrastructure. This was not just a technical experiment but a solution that provided to be stable and easy to manage, making it suitable for small firms. The process was developed step by step, combining components, and as a result integrated environment was achieved, it can complete all tasks: application deployment, network configuration, maintenance. It began with the foundation: first to configure virtual private server, access configuration via SSH key. The next step was to install Coolify, for visually manage and configure applications related tools. When Coolify was installed, two task was started at the same time: first one was to make Coolify available through the domain, and second was to create secure private network to hide admin area from the internet, it was completed with Tailscale and CoreDNS was used to create name resolutions within the private network. To verify the entire system, the Wordpress was deployed. At this stage Coolify demonstrated its potential as almost no manual configuration was required. A template was selected, and the platform automatically deployed the application with all necessary components. The only task was to configure site access. The domain was registered in two locations: Cloudflare to ensure public access and it was integrated into the CoreDNS. This configuration produced the result the blog becomes equally accessible to external and internal access. The practical implementation confirmed the main idea of this thesis: in today's technological landscape, there exist affordable open-source tools that enable small firms to independently configure and maintain their complete infrastructure. As a result, companies can deploy their cloud services without falling into dependence on a specific provider.

5 EVALUATION

The practical work revealed a gap between theoretical knowledge and practical tasks. The simple installation required close attention to details that affected overall system. For instance, it becomes clear that DNS changes are never applied automatically and require time for propagation. Unexpected issues also arose, the need to manually restart services that failed to apply new changes. Each of these situations clearly demonstrated that understanding of infrastructure only comes through practical experience. Abstract concepts remain theoretical until a critical container suddenly fails. The testing confirmed that the automation provided by Coolify can save time during deployment, however it also becomes evident that it is not a universal solution, as its effectiveness depends on the administrator's knowledge. Individuals without IT expertise are unlikely to manage independently and will require assistance as soon as issues arise. The process of configuring and testing this infrastructure demonstrates that the ability to work with real servers and analyze system logs to identify the problems is necessary. it presents approach to troubleshooting: first identifying the problem, then consulting documentation, testing solutions and documenting successful outcomes.

It can be stated that the implementation was successful, however, during the practical work several challenges and limitations arose. First limitation is the scale of the server; the entire architecture was deployed on a single virtual server. While this setup was sufficient for testing purposes and for documenting the proof of concept, it cannot fully represent how the system would perform in a real environment. If it had been possible to operate several servers at the same time, combine them into a cluster, and configure load balancing between them, and separate service, it could show actual performance and system resilience and fault tolerance. Additionally, the server had a fixed amount of hardware resources, which made it impossible to evaluate how Coolify would perform under high load conditions or how effectively the scaling mechanisms would respond when rapid resource expansion was required.

The second aspect was the time required for maintenance. Coolify proved capable of automating a substantial portion of routine tasks, it still required manual intervention. This becomes apparent after installing certain updates or during testing phases. For instance, there were cases where individual containers had to be manually restarted to apply new changes. Or the SSL certificate renewal process had to be initiated manually. These observations led to a conclusion: at its current stage of development,

the platform cannot yet be regarded as a maintenance-free, that can be configured once and left unattended.

This study functioned more as a technical demonstration of what can be achieved rather than a comprehensive evaluation. All conclusions and results were achieved from an experiment conducted within a fully controlled environment. Consequently, they cannot be universally applied to all small firms without adaptation. Also, it is important to note that the work was considered by two main factors: the limited time and the author's own knowledge, which had to be expanded continuously throughout the process. Each new technology implemented, whether Coolify, Tailscale or CoreDNS, required a period of familiarization. Although documentation and community forums proved helpful, they did not always provide comprehensive answers to specific issues. In many cases solutions had to be found through experiments and testing.

6 CONCLUSIONS

This thesis aimed to determine that small firms can rely on a self-hosted infrastructure built with open-source tools. By combining Coolify for deployment and Tailscale for secure networking, the project moved beyond theoretical discussion and demonstrated a functional implementation on a virtual private server. The results confirmed that it is possible to build a system that remains both stable and secure without relying on commercial cloud providers. Coolify managed containerized applications effectively, while Tailscale provided a security layer by enabling private access and closing all public ports. WordPress used as a test case, operated smoothly with automated SSL certificate, and CoreDNS further enhanced the setup by separating public and private DNS zones. These components demonstrated how containerization and automation improve independence from vendor control.

The thesis gives not just practical success but also educational knowledge. The server configuration and resolving network problems and working with a real environment provided an understanding the just theoretical part could not provide. It showed DevOps principles such as automation; security be design and continuous improvement. This work confirmed that open-source software represents a cost-effective solution for small firms, enabling them to create independent infrastructure without dependencies of enterprise licenses. However, the work also underscored that automation operations cannot fully replace human expertise. Tools such as Coolify eliminate repetitive tasks yet still require technical knowledge and regular maintenance. This project validated open source as a viable alternative for small-scale infrastructures.

REFERENCES

Amazon Web Services. 2024. *What is a Virtual Private Server (VPS)?.* Available at: https://aws.amazon.com/what-is/vps/ Accessed 19.2.2026.

Bell, J. & Castro, H. 2022. *Containerization and Orchestration in Cloud Computing.* Available at: https://www.researchgate.net/publication/387958990_Containerization_and_Orchestration_in_Cloud_Computing Accessed 19.9.2025.

Brandao, P.R. 2018. *Cloud Computing: Fundamentals.* International Journal of Computer Science and Technology, 9(2), pp. 87–90. Available at: https://www.researchgate.net/publication/327546112_Cloud_Computing_Fundamentals Accessed 19.9.2025.

Cloudflare Docs 2025. *What is the Secure Shell (SSH) protocol?* Available at: https://www.cloudflare.com/learning/access-management/what-is-ssh/ Accessed 6.10.2025.

Coolify Docs 2025. *Introduction to Coolify.* Available at: https://coolify.io/docs/get-started/introduction Accessed 3.2.2026

DediRock 2025. *The Environmental Benefits of Using VPS Hosting.* Available at: https://dedirock.com/blog/the-environmental-benefits-of-using-vps-hosting/ Accessed 3.2.2026

DigitalOcean 2025. *About DigitalOcean.* Available at: https://www.digitalocean.com/about Accessed 5.10.2025.

Dubendorfer, D. 2025. *AWS vs Azure vs Google: Cloud Services Comparison. Varonis.* Available at: https://www.varonis.com/blog/aws-vs-azure-vs-google Accessed 13.9.2025.

Forward Team 2025. *On-Site to Cloud: Why Businesses Choose VPS Over On-Site Servers.* Air Link Data Center. Available at: https://airlinkdc.com/on-site-to-cloud-why-businesses-choose-vps-over-on-site-servers/ Accessed 16.9.2025.

Kalsi, H.S. 2024. *Understanding Open-Source and Is Open-Source Better Than Closed Source?* Medium. Available at: https://medium.com/@harrpreet/understanding-open-source-and-is-open-source-better-than-closed-source-4f7cc8fd9fab Accessed 15.9.2025.

Madison, M.J. 2005. *The Legitimacy of Open Source and Other Software Licenses.* University of Pittsburgh School of Law. Available at: https://www.researchgate.net/publication/237467944_The_Legitimacy_of_Open_Source_and_Other_Software_Licenses Accessed 20.9.2025.

Mujalli, A., Wani, M.J.G., Almgrashi, A., Khormi, T. & Qahtani, M. 2024. *Investigating the factors affecting the adoption of cloud accounting in Saudi Arabia's small and medium-sized enterprises (SMEs). Journal of Innovation and Knowledge*, Elsevier. Available at: https://www.researchgate.net/publication/381305838_Investigating_the_factors_affecting_the_adoption_of_cloud_accounting_in_Saudi_Arabia's_small_and_medium-sized_enterprises_SMEs Accessed 3.2.2026.

Opara-Martins, J., Sahandi, R. & Tian, F. 2014. *Critical review of vendor lock-in and its impact on adoption of cloud computing*. In: International Conference on Information Society (i-Society 2014).

IEEE. Available at: https://www.researchgate.net/publication/272015526_Critical_Review_of_Vendor_Lock-in_and_its_Impact_on_Adoption_of_Cloud_Computing Accessed 13.9.2025.

Red Hat. 2024. *What is virtualization?.* Available at: https://www.redhat.com/en/topics/virtualization/what-is-virtualization Accessed 19.2.2026.

SkySilk 2024. *Coolify: A Modern VPS Hosting Solution for the Cloud.* SkySilk Blog, 26 November 2024. Available at: https://www.skysilk.com/blog/coolify-a-modern-vps-hosting-solution-for-the-cloud/ Accessed 15.10.2025.

Susnjara, S. & Smalley, I. 2024. *What is a hypervisor?.* Available at: https://www.ibm.com/think/topics/hypervisors Accessed 19.2.2026.

Susnjara, S. & Smalley, I. n.d. What is containerization? IBM Think. Available at: https://www.ibm.com/think/topics/containerization Accessed 3.1.2026.

Tailscale Docs 2025. *What is Tailscale.* Available at: https://tailscale.com/kb/1151/what-is-tailscale Accessed 6.10.2025.

Yasar, K., Chai, W. & Bigelow, S. J. 2025. *What is cloud computing? Types, examples and benefits.* Available at: https://www.techtarget.com/searchcloudcomputing/definition/cloud-computing Accessed 19.2.2026.

APPENDIX 1

SETUP SCRIPTS

This appendix presents the key setup scripts used to initialize and configure the project environment. The scripts automate the installation and connection of essential open-source tools.

```
curl -fsSL https://cdn.coollabs.io/coolify/install.sh | sudo bash
```

Code 1. Coolify install script

```
ssh root@cp.serhii-didenko.me
```

Code 2. SSH login to the server

```
curl -fsSL https://tailscale.com/install.sh | sh
```

Code 3. Tailscale install script

CONFIGURATION FILES

This appendix contains the configuration files used to define network resolution and service orchestration within the self-hosted environment.

```
serhii-didenko.me {
   bind 100.88.58.77
   log
   errors
   hosts {
      100.88.58.77            cp.serhii-didenko.me # coolify panel on ip v4
      fd7a:115c:a1e0::4f01:3a50 cp.serhii-didenko.me # coolify panel on ip v6
      10.0.1.6                coolify.serhii-didenko.me # an example of a service
      10.0.1.5                db.serhii-didenko.me # an example of a service
      10.0.1.3                redis.serhii-didenko.me # an example of a service
      10.0.0.2                sentinel.serhii-didenko.me # an example of a service
      fallthrough
   }
   forward . 1.1.1.1 8.8.8.8
   cache 120
}

. {
   bind 100.88.58.77
   forward . 1.1.1.1 8.8.8.8
   log
   errors
}
```

Code 4. coreDNS config file

CONFIGURATION FILES

```
version: "3.9"
services:
  coredns:
    image: coredns/coredns:latest
    container_name: coredns
    restart: always
    network_mode: host
    volumes:
      - ./Corefile:/Corefile:ro
    command: -conf /Corefile
```

Code 5. Docker-compose config file for the coreDNS service

```
hosts {
    100.88.58.77            blog.serhii-didenko.me
    fallthrough
}
```

Code 6. Config for new subdomain in the coreDNS service