Project Report

University School of Automation & Robotics
GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY
East Delhi, Surajmal Vihar Delhi - 110092

Cloud, Dew, Edge, and Fog Computing Project Report
ARM 305

Deployment and Management of Apache Web Servers on Ubuntu

Sarthak Maurya [146]
Mayur Raj [135]
Honey Singh [124]
Eshaan Gupta [145]
Durga Sharma [120]
Ayush Kumar [102]

Under the Supervision of **Dr. Abhishek Singh**

Declaration

We hereby declare that the Lab Project Report entitled "Deployment and Management of Apache Web Servers on Ubuntu" is an authentic record of work completed in partial fulfillment of the requirements for the Lab Project at University School of Automation and Robotics

Date:

Sarthak Maurya	**[146]**
Eshaan Gupta	**[145]**
Ayush Kumar	**[102]**
Mayur Raj	**[135]**
Honey Singh	**[124]**
Durga Sharma	**[120]**

Date: **Dr. Abhishek Singh**

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our mentor, Dr. Abhishek Singh, for his invaluable guidance, encouragement, and insights throughout our project, **Deployment and Management of Apache Web Servers on Ubuntu**. His expertise and support were instrumental in helping us navigate challenges and deepen our understanding of the subject.

We also extend our heartfelt thanks to our peers and colleagues for their collaboration and assistance, making this project a truly educational and rewarding team effort. Working together as a group of six, we have gained invaluable skills and knowledge that will undoubtedly contribute to our professional growth.

Finally, we would like to express our deepest appreciation to our families for their unwavering support and belief in us, motivating us to strive for success. This project has been a pivotal experience in our journey, and we are grateful to everyone who contributed to its successful completion.

University School of Automation and Robotics

The University School of Automation and Robotics (USAR) at Guru Gobind Singh Indraprastha University (GGSIPU) is dedicated to advancing education and research in automation, robotics, and artificial intelligence. Situated at the East Delhi Campus in Surajmal Vihar, Delhi, USAR represents a forward-looking approach to technical education, aligning with the latest industry advancements in robotics, automation, and intelligent systems.

USAR offers a wide array of undergraduate, postgraduate, and doctoral programs designed to equip students with comprehensive knowledge in engineering, robotics, machine learning, and automation. The school's curriculum is tailored to integrate theoretical foundations with hands-on practical training, fostering a research-oriented learning environment. With state-of-the-art laboratories, high-end computing resources, and a collaborative research culture, USAR is committed to producing highly skilled professionals who can contribute meaningfully to the fields of robotics and automation.

Beyond academics, USAR emphasizes holistic student development by encouraging participation in technical events, hackathons, and competitions. Collaborations with industry leaders, research institutions, and other universities provide students with exposure to real-world applications and the latest technological innovations. As a leader in automation and robotics education, USAR continually inspires students to push the boundaries of technology and pursue excellence in their fields.

List of Figures

Abstract

This report explores the deployment of an Apache web server on Ubuntu, examining Apache's role in web hosting. It summarizes the objective of establishing a secure, reliable web server and covers the configurations, setup steps, and tools needed to deploy a functional server environment. The project emphasizes the importance of virtual hosting to serve multiple websites from one server, the role of firewalls in securing web servers, and the relevance of Apache's modular design. Through hands-on practice, team members gained foundational skills in server management and optimization, creating a scalable hosting environment for various applications.

Introduction

In the modern digital era, web servers are the backbone of online experiences, enabling dynamic, interactive content delivery over the internet. This project focuses on configuring a reliable web server using Apache on an Ubuntu machine. Apache has established itself as a cornerstone in web hosting, providing a flexible, open-source solution for both small and large applications. The project's aim is to understand the technical foundation required to deploy a scalable server, allowing seamless delivery of web applications. Various configurations were implemented to optimize performance and security, ensuring the server environment meets industry standards and operational requirements. By working with Apache, project members acquired valuable skills in server setup, management, and application hosting, equipping them for future roles in IT infrastructure and web management.

Apache

Apache is one of the most widely used open-source web server platforms, designed to deliver web content over the HTTP protocol. Known for its stability, Apache has evolved with contributions from the global open-source community, resulting in a feature-rich, highly flexible server. Apache supports modular functionalities, meaning features can be added or removed based on specific requirements. For example, Apache's modules enable SSL/TLS for encrypted communication, caching to improve speed, and integration with languages like PHP and Python to serve dynamic content. As a central component of the LAMP (Linux, Apache, MySQL, PHP) stack, Apache is also compatible with various operating systems, enhancing its usability across platforms. With dynamic module loading, comprehensive security options, and extensive support for frameworks and tools, Apache's adaptability has cemented its role in modern web hosting.

Server

A server is a powerful computer system that stores, processes, and delivers data to other devices, or clients, over a network. Specifically, a web server stores and serves web content, responding to requests from clients (such as web browsers) to deliver files, applications, or information. Servers can be classified into various types based on their functionality. Application servers process the back-end logic of applications, enabling data interactions, while database servers manage and store data. A web server, like Apache, primarily handles HTTP requests, making it accessible through the web. By supporting interactions with other servers (e.g., database servers), web servers allow users to interact with data in real-time, forming the backbone of internet services. In this project, Apache is used as the web server to manage and deliver content to users, demonstrating the server's capability to handle multiple concurrent requests and its relevance in scalable application deployment.

Hosting

Hosting refers to providing space on a server to store and manage data, applications, and websites, making them accessible to users over the internet. Hosting is typically offered as a service, with providers offering a range of solutions from shared hosting to dedicated servers. Shared hosting involves multiple websites sharing the same server resources, while VPS (Virtual Private Server) hosting offers a more isolated environment with dedicated resources within a shared server. Dedicated hosting involves an entire server being allocated to a single client, providing exclusive resources and control. Cloud hosting is a scalable option where resources are allocated dynamically across multiple servers, offering flexibility and redundancy. Apache web hosting uses Apache as the primary server, allowing it to manage static and dynamic content. Apache’s compatibility with various technologies and protocols, including SSL/TLS for secure connections, enhances its functionality for secure, efficient web hosting.

Apache Web Hosting and Why Apache is Preferred

Apache web hosting is the practice of using Apache software to manage and deliver website content. Apache’s reputation for reliability, flexibility, and extensive community support has made it a preferred choice in the web hosting industry. Apache’s modular nature allows it to serve diverse needs, with features such as mod_php and mod_ssl enabling dynamic content delivery and secure connections. As an open-source solution, Apache has been continuously updated and improved, leading to a rich set of features and high stability. Additionally, Apache’s compatibility with Linux and integration into the LAMP stack make it ideal for web applications, offering seamless compatibility with popular database management systems like MySQL and development languages such as PHP. These advantages make Apache well-suited for both small-scale and enterprise-level hosting, providing an adaptable solution for various web hosting scenarios.

Relation to Cloud Computing

Cloud computing has transformed traditional hosting by offering scalable, flexible infrastructure that allows applications to adapt to fluctuating demands. Apache's compatibility with cloud environments makes it well-suited for modern cloud-based hosting solutions. When deployed in the cloud, Apache can scale horizontally, enabling the addition of multiple instances to handle high traffic. Cloud-based load balancers and auto-scaling features allow Apache to serve large-scale applications without overloading the server. Apache can also integrate with cloud storage solutions, providing seamless data access across distributed environments. By deploying Apache on a cloud platform, organizations can reduce infrastructure costs, scale services dynamically, and manage server resources more efficiently.

Infrastructure Classification: IaaS (Infrastructure as a Service)

This project, focused on the deployment and management of an Apache web server on an Ubuntu operating system, aligns with the Infrastructure as a Service (IaaS) model. IaaS provides users with virtualized computing resources over the internet, allowing them to control the operating system, network configurations, and installed software. In this setup, we provisioned an environment where we directly managed the Ubuntu OS, installed and configured the Apache server, and established server security and performance settings. Each aspect of the server's deployment, from firewall configurations to virtual host setup, required hands-on management at the OS and software level.

This type of setup is characteristic of IaaS, where the infrastructure provider offers the foundational computing resources, such as virtual machines, networking, and storage, and the user takes full responsibility for managing the operating system, server applications, and middleware. For instance, if hosted on a platform like AWS EC2 or Google Compute Engine, this project would fall under IaaS, as it involves configuring the server environment from the ground up.

In contrast, a Platform as a Service (PaaS) model would abstract these infrastructure details, offering a managed environment where users deploy applications without handling OS or server configurations. PaaS services, such as Heroku or Google App

Engine, provide an environment where much of the underlying infrastructure is managed by the provider, leaving users to focus solely on application development. In this project, however, we configured both the OS and the Apache server directly, which classifies it within the IaaS model rather than PaaS.

This hands-on approach enabled us to gain practical skills in server management and configurations, reinforcing foundational knowledge of infrastructure setup and maintenance as part of an IaaS deployment.

Comparison with SaaS (Software as a Service)

Unlike the IaaS approach taken in this project, which involves managing and configuring server infrastructure directly, Software as a Service (SaaS) provides a fully managed application that end-users can access over the internet, without needing to handle any aspect of the underlying infrastructure, server setup, or software maintenance. SaaS applications are hosted and maintained by a third-party provider, allowing users to simply log in and use the software. Examples of SaaS include platforms like Google Workspace, Salesforce, and Microsoft 365, where the application is ready-to-use, and users do not have to worry about updates, hosting, security patches, or server configurations.

In this project, we worked directly on the server layer by deploying an Apache server on Ubuntu, setting up firewall rules, and configuring virtual hosts, which places this work within the IaaS model. A SaaS approach would have meant using a fully developed web hosting or website management application, where users could upload content or applications without needing to install or configure the server.

While SaaS is ideal for end-users looking for simplicity and minimal setup, IaaS offers more control and flexibility for developers and administrators who need direct access to the server environment. Therefore, this project differs significantly from SaaS by requiring direct interaction with the server infrastructure, thus providing valuable insights into the foundational aspects of server and application management.

Project Prerequisites

Setting up an Apache server on Ubuntu requires specific hardware and software configurations to ensure a stable and secure environment capable of handling web traffic efficiently. The following prerequisites provide a foundation for establishing a reliable server infrastructure, focusing on hardware and software compatibility, security best practices, and access control.

To deploy Apache on Ubuntu, certain hardware and software requirements must be met to ensure optimal performance. At a minimum, the server hardware should include a 1 GHz processor, 1 GB of RAM, and 10 GB of storage to support the Ubuntu operating system and Apache services effectively. These specifications enable basic web hosting, but higher resource allocations may be necessary depending on the scale of hosted applications and anticipated traffic. For instance, a server intended for high-traffic websites or multiple domains should ideally have 2+ GHz processing power, 4 GB or more of RAM, and at least 20 GB of storage for logs, data, and additional software installations. On the software side, the server should run Ubuntu 22.04 LTS (Long-Term Support), which provides an up-to-date Linux distribution with extensive support for server environments. Apache HTTP Server is required for web hosting functionality, while OpenSSH is recommended for secure remote access. With these minimum requirements met, the server will be well-equipped to handle basic web hosting needs, with room to scale based on additional resource availability.

The choice of Ubuntu 22.04 LTS as the operating system offers several benefits that make it ideal for server deployment. Ubuntu 22.04 is known for its stability, as it is built on a foundation of long-term support, with updates and security patches provided for at least five years. This LTS version is particularly valuable for web servers that need to operate reliably without frequent system upgrades, minimizing downtime and maintenance. Additionally, Ubuntu is one of the most widely supported Linux distributions, with compatibility across a broad range of server environments and cloud platforms, including AWS, Google Cloud, and Microsoft Azure. This compatibility ensures that the server can be easily deployed in various infrastructure setups, including on-premise and cloud environments. Ubuntu 22.04 also boasts a large, active community that provides extensive resources, tutorials, and troubleshooting support, which can be invaluable for maintaining and configuring the server. Its compatibility with Apache is seamless, as both are widely used together, allowing for streamlined setup and configuration with many resources available to guide users through the process.

Another critical security measure in this setup is configuring a non-root user for administrative tasks. By default, root access provides unrestricted control over the system, which can pose significant security risks if an unauthorized user gains access. Creating a non-root user allows for delegation of necessary administrative privileges without granting full access to sensitive system files or configuration settings. This approach minimizes the risk of accidental or malicious modifications to critical system components, reducing the likelihood of server compromise. In this project, we create a non-root user specifically for managing Apache and other server tasks, ensuring that only essential permissions are granted. This user can perform administrative actions through the use of `sudo` (superuser do) privileges, which temporarily elevates permissions as needed. By limiting root access, we adopt a security best practice that restricts access and provides an added layer of protection for the server environment.

To further safeguard the server, it is essential to set up a firewall using UFW (Uncomplicated Firewall), which serves as a network traffic filter that allows or denies specific types of connections based on predefined rules. A firewall acts as the first line of defense, protecting the server from unauthorized access, malicious attacks, and unwanted traffic. In this project, UFW is configured to permit only HTTP and HTTPS traffic, which are essential for web server operations, while blocking all other incoming connections by default. This selective filtering prevents potentially harmful traffic from reaching the server, reducing the risk of cyberattacks or unauthorized access. To set up UFW, we begin by listing available application profiles using the command `sudo ufw app list` to identify the necessary Apache profiles. Once identified, we enable HTTP and HTTPS traffic by allowing the ‘Apache’ profile through UFW using the command `sudo ufw allow 'Apache'`. Finally, we verify the firewall’s active status with `sudo ufw status` to ensure that the rules have been applied successfully. Configuring UFW in this manner helps maintain a secure server environment by tightly controlling network access, further bolstering the server's resilience against potential threats.

These prerequisites—ensuring appropriate hardware and software requirements, choosing Ubuntu 22.04 LTS for stability, setting up a non-root user for secure administration, and configuring UFW as a protective firewall—create a solid foundation for deploying and managing an Apache server. Together, they enable a secure, scalable, and efficient web hosting environment capable of supporting various applications and handling server demands with confidence.

Deployment

Deploying Apache on Ubuntu involves a structured setup process to establish a secure and manageable environment. Using Ubuntu 22.04 brings long-term support and reliability, ensuring that the server environment is backed by regular security and maintenance updates. Creating a non-root user for administrative purposes is a vital step in safeguarding the server, as it limits access to sensitive configurations. Configuring UFW adds another layer of security by allowing only authorized HTTP and HTTPS traffic to reach the server, protecting against unauthorized access and potential vulnerabilities. By structuring each component of the deployment process, including the selection of the operating system, firewall configuration, and user management, this approach ensures a scalable, resilient setup that can adapt to evolving demands and applications.

Step-by-Step Setup and Management

The installation process begins with updating the package index, followed by installing Apache. Once Apache is installed, configuring the firewall involves listing available profiles, authorizing HTTP traffic, and verifying that the firewall is active. After verifying that Apache is running properly, the virtual hosts setup allows multiple websites to be hosted on a single server. This process involves creating specific directories for each domain, setting permissions, and configuring virtual host files. Managing Apache involves commands to start, stop, and restart the server, as well as enabling or disabling auto-start. Key directories and logs are vital for ongoing server management, as they allow administrators to monitor the server's status and troubleshoot any issues. This structured approach ensures that the server is fully prepared to handle web traffic in a secure, efficient manner.

The deployment of Apache on an Ubuntu server follows a structured, step-by-step process to ensure a reliable, secure, and fully functional web server environment. Each stage is designed to set up, configure, and manage Apache effectively, from installation through to managing multiple domains with virtual hosts. The following sections detail each deployment step, covering the purpose, rationale, and commands used, along with insights on troubleshooting.

Step 1: Installing Apache

To begin deploying Apache, we first install it on our Ubuntu machine. What we aim to accomplish in this step is the foundational setup of the Apache web server on the system. Why this step is crucial is because Apache serves as the core software responsible for handling and responding to client requests over HTTP/HTTPS, making it essential for web hosting. How to proceed involves updating the system's package index to ensure all available software versions are current, followed by using the command to install Apache.

The exact commands are as follows:

1. Update the package index to ensure access to the latest software:
   sudo apt update

2. Install Apache on the server:
   sudo apt install apache2

Once installed, Apache starts automatically, and the server is now equipped with the Apache service. To verify the installation, you can open a web browser and navigate to `http://<your_server_ip>`, which should display Apache's default web page. If this page is visible, it confirms that Apache is installed correctly and running. If Apache doesn't start automatically, or if there's no response, it may be due to system compatibility or network restrictions. Restarting the Apache service with `sudo systemctl restart apache2` can often resolve these issues.

Step 2: Configuring the Firewall

After installing Apache, the next step is to secure server access by configuring the firewall using UFW (Uncomplicated Firewall). What we do here is define rules for managing incoming and outgoing traffic, specifically allowing only trusted connections to reach the Apache server. Why this step is necessary is to protect the server from unauthorized access and to filter the types of traffic allowed, reducing exposure to potential security threats. How we configure UFW involves listing the available application profiles, enabling HTTP and HTTPS traffic, and verifying the firewall's status.

The configuration commands include:

1. List available application profiles to confirm Apache profiles are available:
sudo ufw app list

2. Allow HTTP traffic through the firewall:
sudo ufw allow 'Apache'

3. Verify the firewall's status to ensure rules are applied correctly:
sudo ufw status

Once these rules are configured, only HTTP and HTTPS traffic will be allowed to access Apache, thereby securing the server. If UFW reports any errors, restarting the firewall with `sudo ufw disable` followed by `sudo ufw enable` can help reset the configuration. If issues persist, reviewing `/etc/ufw/ufw.conf` for any misconfigurations may be necessary.

Step 3: Verifying Installation

With Apache installed and the firewall configured, the next step is verifying the installation to ensure Apache is active and accessible. What we achieve here is confirmation that the server is operational, with Apache actively listening for client requests. Why this step is crucial is that it enables quick identification of any issues early in the deployment process, ensuring that the server is set up correctly. How we proceed involves checking the Apache service status and testing the connection through a browser.

1. Check the status of the Apache service:

```
sudo systemctl status apache2
```

If Apache is running, the service status will display as “active.” You can also test the setup by entering `http://<your_server_ip>` in a browser, which should display Apache’s default page. If Apache isn’t running, restarting it with `sudo systemctl restart apache2` can resolve the issue. If connectivity issues arise, verifying firewall settings to ensure Apache traffic is allowed is also a good troubleshooting step.

Step 4: Setting Up Virtual Hosts

After verifying Apache’s functionality, the next phase involves setting up virtual hosts to allow the server to host multiple domains. What we aim to accomplish here is the configuration of separate environments for each website or application on the same server. Why this setup is beneficial is because it maximizes server resources, allowing multiple sites to be managed efficiently without requiring additional hardware. How we achieve this involves creating directories for each domain, setting permissions, and configuring individual virtual host files.

The steps include:

1. Create a directory for each domain:
sudo mkdir -p /var/www/your_domain

2. Set directory permissions and ownership:
 sudo chown -R $USER:$USER /var/www/your_domain

3. Configure the virtual host file in
`/etc/apache2/sites-available/your_domain.conf`: plaintext

```
<VirtualHost *:80>
    ServerAdmin admin@your_domain
    ServerName your_domain
    ServerAlias www.your_domain
    DocumentRoot /var/www/your_domain
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

After configuring the virtual host, enable it using `sudo a2ensite your_domain.conf` and restart Apache with `sudo systemctl restart apache2`. To confirm that each site is accessible, navigate to the specific domain name in a browser. If the configuration doesn't work as expected, ensuring the domain name resolves correctly in DNS or checking for syntax errors in the configuration file can help troubleshoot.

Step 5: Managing Apache

To keep Apache functioning optimally, it's essential to understand the management commands for controlling the server. What we do in this step is learn the primary commands for starting, stopping, restarting, and managing Apache's autostart settings. Why these commands are important is that they enable routine maintenance and ensure that Apache responds reliably to changes in configuration or traffic patterns. How we use these commands involves simple but crucial terminal commands.

The management commands are as follows:

1. Start, stop, or restart Apache:

```
sudo systemctl start apache2
sudo systemctl stop apache2
sudo systemctl restart apache2
```

2. Enable or disable Apache autostart:

```
sudo systemctl enable apache2
sudo systemctl disable apache2
```

By using these commands, administrators can ensure that Apache runs when needed and is shut down during maintenance. If Apache fails to start, checking the status with `sudo systemctl status apache2` can provide error messages that guide troubleshooting.

Step 6: Key Directories and Logs

The final step in managing Apache is understanding the key directories and logs essential for monitoring and troubleshooting. What we cover here are the primary configuration and log file locations used for managing Apache. Why these files are significant is because they provide a record of server activity and errors, which are invaluable for diagnosing issues. How to locate and interpret these files involves knowing the directory paths for configuration and logs.

1. Configuration Files:
   - Main configuration: `/etc/apache2/apache2.conf`
   - Virtual hosts: `/etc/apache2/sites-available/`

2. Logs:
   - Access log: `/var/log/apache2/access.log`
   - Error log: `/var/log/apache2/error.log`

The configuration files contain settings that govern Apache's behavior, while the logs record all access requests and errors encountered by the server. Reviewing the error log in particular can help troubleshoot configuration issues or detect potential security threats. Regularly monitoring these logs allows administrators to respond quickly to server issues, ensuring optimal performance.

Advantages of Apache Web Hosting

Apache offers significant advantages as a web hosting solution, including high performance, flexibility, modularity, compatibility, and strong community support, making it one of the most widely adopted web servers worldwide. Apache is known for its reliable performance, capable of handling high volumes of web traffic efficiently. Its architecture allows for the management of concurrent connections without sacrificing speed, which is essential for maintaining a responsive web hosting environment. Additionally, Apache's flexibility supports a wide range of configurations, allowing administrators to tailor the server's behavior to specific application needs. Whether hosting static websites, serving dynamic applications, or managing multiple sites on a single server, Apache's adaptable configuration options enable it to handle diverse requirements, making it a versatile choice for web hosting across various use cases.

One of Apache's standout features is its modular architecture, which allows for dynamic module loading. This modularity means that administrators can enable or disable specific functionalities by adding or removing modules, giving them precise control over the server's features and resource usage. Modules such as `mod_ssl` for secure connections, `mod_rewrite` for URL redirection, and `mod_proxy` for load balancing can be integrated as needed without altering the server's core functionality. Dynamic module loading provides not only flexibility but also efficiency, as administrators can activate only the necessary features, thereby optimizing server performance. This modular approach makes Apache scalable and customizable, able to meet the unique requirements of various applications and environments, which is especially beneficial for growing businesses that may need to scale their server capabilities over time.

Compatibility is another advantage that makes Apache a popular choice for web hosting. Apache’s support for the Linux, Apache, MySQL, PHP (LAMP) stack has solidified its position as a foundational technology for web development and hosting. Additionally, it is compatible with other operating systems, including Windows and macOS, allowing it to fit into a variety of infrastructure setups. Apache’s support for SSL/TLS protocols is crucial for secure connections, especially in an era where data privacy and security are top priorities for businesses and users alike. With SSL/TLS integration, Apache ensures that data transmitted between the server and clients remains encrypted, protecting sensitive information from potential threats. This compatibility with various platforms and security protocols enables Apache to be deployed in nearly any hosting environment, from small personal projects to large-scale enterprise applications, further contributing to its widespread adoption.

Another significant advantage of Apache is its strong community support. As an open-source project, Apache has benefited from decades of contributions from a global community of developers, administrators, and users who continuously improve its features, fix bugs, and enhance security. This active community has contributed to a wealth of documentation, tutorials, and resources, making it easier for new users to learn and troubleshoot Apache. Additionally, open-source contributions ensure that Apache remains at the cutting edge of web server technology, with regular updates and improvements that keep it relevant in an ever-evolving technological landscape. Community-driven support also means that Apache has a high level of transparency, with its source code publicly available for review and modification. This open-source nature not only fosters trust but also ensures that organizations can rely on Apache as a secure, adaptable, and well-maintained web server solution.

In summary, Apache’s performance, flexibility, modularity, compatibility, and community support make it a highly advantageous choice for web hosting. Its ability to handle high traffic volumes, adapt through modular configurations, integrate with various platforms, and secure data transmission through SSL/TLS, combined with the backing of a strong open-source community, establishes Apache as one of the most reliable and scalable web servers available today.

Conclusion

The project provided valuable insights into deploying and managing Apache as a web server on Ubuntu. By configuring virtual hosts, we demonstrated Apache’s capability to manage multiple domains efficiently, maximizing resource usage on a single machine. Analyzing access and error logs offered practical skills in troubleshooting, highlighting the importance of monitoring for continuous server health. Each aspect of the project contributed to a foundational understanding of server deployment and management, equipping us for future exploration into advanced configurations, such as load balancing, caching, and scaling Apache in cloud environments. Through this project, we developed a comprehensive understanding of web server hosting, laying a solid base for further work in IT infrastructure.

References

1. DigitalOcean. *Apache Web Server Deployment Guide*. DigitalOcean provides extensive tutorials on deploying and managing Apache Web Servers, which served as a foundational resource for setting up our web servers. Available at: https://www.digitalocean.com/community/tutorials
2. VirtualBox. *User Manual and Virtual Machine Setup*. The VirtualBox platform facilitated our testing environment, allowing us to virtualize Ubuntu instances effectively. The official documentation and community forums were instrumental in configuring our virtual machines. Available at: https://www.virtualbox.org/manual
3. Ubuntu. *Ubuntu Server Documentation*. The Ubuntu documentation provided detailed guidance on server setup, networking, and package management, helping ensure smooth configuration of our server environment. Available at: https://ubuntu.com/server/docs
4. Apache. *Apache HTTP Server Documentation*. Apache's official documentation was an invaluable reference for configuring and managing the server, including performance tuning, security practices, and module installation. Available at: https://httpd.apache.org/docs