# Table of Contents

[TABLE OF CONTENTS i](#_Toc172390787)

[INTRODUCTION 1](#_Toc172390788)

[I. PRESENTATION 2](#_Toc172390789)

[A. DEFINITION AND IMPORTANCE OF A REMOTE LOCAL SERVER 2](#_Toc172390790)

[B. PROBLEM: FUNCTIONING AND CHALLENGES 3](#_Toc172390791)

[C. OBJECTIVES OF THE PRESENTATION 4](#_Toc172390792)

[II. SETUP 5](#_Toc172390793)

[REQUIREMENTS AND SETUP 5](#_Toc172390794)

[1. Server Hardware and OS Installation 5](#_Toc172390795)

[2. Network Configurations 5](#_Toc172390796)

[3. Remote Access 5](#_Toc172390797)

[4. File Sharing 6](#_Toc172390798)

[5. Virtual Private Network 7](#_Toc172390799)

[6. Firewall 8](#_Toc172390800)

[7. Implementing Fail2Ban for Enhanced Server Security 8](#_Toc172390801)

[8. Monitoring 9](#_Toc172390802)

[9. Data Backup 10](#_Toc172390803)

[IMPLEMENTATION 11](#_Toc172390804)

[III. IMPLEMENTATION STEPS 11](#_Toc172390805)

[1. Server Hardware and OS Installation 11](#_Toc172390806)

[2. Network Configurations 12](#_Toc172390807)

[3. Remote Access 13](#_Toc172390808)

[4. File Sharing 19](#_Toc172390809)

[5. Virtual Private Network 26](#_Toc172390810)

[6. Firewall 28](#_Toc172390811)

[7. Implementing Fail2Ban for Enhanced Server Security 32](#_Toc172390812)

[8. Monitoring 32](#_Toc172390813)

[9. Data Backup 37](#_Toc172390814)

[10. Troubleshooting 38](#_Toc172390815)

[CONCLUSION 39](#_Toc172390816)

In the rapidly evolving digital landscape, the establishment of a secure local server, complemented by a robust remote management client, is paramount for maintaining the integrity and confidentiality of data. This report delves into the meticulous process of designing, installing, and configuring a secure local server infrastructure, while also integrating a secure remote management client to ensure seamless and secure access for authorized personnel. Through a comprehensive analysis of current security protocols, hardware requirements, and software configurations, this report aims to provide a detailed roadmap for organizations seeking to fortify their network against potential cyber threats. The subsequent sections will outline the critical steps and considerations necessary to achieve a resilient and secure server environment, tailored to meet the specific needs of modern enterprises.

# INTRODUCTION

## PRESENTATION

### DEFINITION AND IMPORTANCE OF A REMOTE LOCAL SERVER

**Definition of a Remote Local Server**

A remote local server refers to a server that is situated within a local network but can be accessed and managed remotely. This type of server combines the advantages of local data control with the flexibility of remote access. In the context of designing, installing, and configuring a secure local server with a secure remote management client, the importance of a remote local server cannot be overstated. It serves as the backbone of network architecture, providing a centralized point where data can be stored, managed, and retrieved efficiently.

The secure remote management client is a critical component that ensures administrators can perform maintenance, updates, and troubleshooting tasks from any location, without the need for physical presence. This not only reduces downtime but also enhances the responsiveness of the IT support structure. The integration of a secure remote management client with a local server ensures that security protocols such as encryption, authentication, and authorization are consistently applied, safeguarding sensitive information against unauthorized access and cyber threats.

In an era where businesses operate across multiple locations and time zones, the ability to manage servers remotely is invaluable. It allows for continuous monitoring and immediate response to incidents, which is essential for maintaining uninterrupted business operations. Furthermore, the deployment of a remote local server with a secure management client aligns with disaster recovery strategies, enabling organizations to quickly restore services in the event of a system failure or other disruptive scenarios.

Overall, the implementation of a remote local server with a secure remote management client is a strategic investment that enhances operational efficiency, fortifies security measures, and provides a scalable solution that can adapt to the evolving needs of a business. It is a testament to the foresight of an organization that prioritizes robust infrastructure and proactive management to support its long-term objectives and resilience in the face of technological advancements and security challenges.

### PROBLEM: FUNCTIONING AND CHALLENGES

In the realm of network administration, the establishment of a secure local server with a remote management client presents a unique set of challenges. The primary concern is ensuring the seamless functioning of the server, which necessitates a robust design capable of withstanding various security threats. The intricacies of installation and configuration also pose significant hurdles, as they require precise coordination and a deep understanding of network protocols and security measures. Furthermore, maintaining the integrity of remote management capabilities is paramount; this involves safeguarding against unauthorized access and ensuring that remote interactions do not compromise the server's security. The balance between accessibility and security is delicate, and the server must be fortified against both external and internal threats. This report will delve into these challenges in detail, proposing solutions to optimize server performance while maintaining stringent security standards.

### OBJECTIVES OF THE PRESENTATION

The primary objective of this presentation is to elucidate the comprehensive process of designing, installing, and configuring a secure local server that is complemented by a robust remote management client. This will encompass the deployment of SSH for secure remote access, ensuring a seamless and encrypted connection for administrators. Additionally, the integration of Samba for efficient network storage solutions will be detailed, highlighting its utility in resource sharing across different operating systems. The presentation will also cover the implementation of advanced firewall protocols to fortify the server's defense against unauthorized access and potential security threats. Furthermore, the utilization of WireGuard for establishing a VPN will be discussed, showcasing its effectiveness in maintaining privacy and security for remote communications. The overarching aim is to present a clear, intuitive, and professional overview of the server's architecture, emphasizing the importance of security measures in today's interconnected digital landscape.

## SETUP

# REQUIREMENTS AND SETUP

To set up a remote local server for secure management and network storage, we will need to follow a series of steps to ensure both functionality and security.

### Server Hardware and OS Installation

The minimum requirements for such a server include a dedicated machine with at least a dual-core processor, 4GB of RAM, and a 500GB hard drive. For the operating system, a Linux distribution such as Ubuntu Server is recommended due to its stability and community support.

The first step is to install the operating system and update it to the latest version.

### Network Configurations

Once the OS is installed, Assign a static IP address to the Server on the network. This ensures that the servers IP address on the network is always known

* It is recommended to set it up on the router’s DHCP settings instead of assigning a static IP in the server’s network configurations.
* This will prevent future issues in case there’s a need to upgrade or change network, or if any devices happen to use the IP address set manually.

### Remote Access

we'll need to configure the SSH service for secure remote access. **Secure Shell (SSH)** is the superior choice for remote access, and here’s why:

Security: SSH encrypts all data exchanged between the client and server, ensuring confidentiality and integrity. It safeguards sensitive information from unauthorized access and interception.

* Authentication: SSH uses public-key cryptography, making it highly resistant to password-based attacks. Public key authentication adds an extra layer of security, reducing the risk of brute force attempts.
* Port Forwarding: SSH allows secure access to services on a remote server through encrypted tunnels. This feature is especially useful for accessing internal resources behind firewalls.

In summary, SSH provides strong encryption, secure authentication, and robust tunneling capabilities, making it ideal for system administration, remote server management, and secure file transfers, set up user accounts with secure authentication and disabling password login to prevent unauthorized access.

### File Sharing

File sharing is a crucial aspect of network management, allowing multiple users to access and share files across different devices. It enhances collaboration, reduces redundancy, and ensures that everyone has access to the most up-to-date information. One of the most popular tools for file sharing in a networked environment is Samba.

Samba is an open-source software suite that provides seamless file and print services to SMB/CIFS clients. It allows for interoperability between Linux/Unix servers and Windows-based clients, making it a versatile choice for mixed-OS environments. Samba uses the SMB (Server Message Block) protocol, which is also known as CIFS (Common Internet File System), to facilitate file sharing and network communication.

To set up Samba for file sharing, we need to install the Samba package on our designated server. Once installed, we configure the Samba server by editing the `smb.conf` file, which is the main configuration file for Samba. This file allows us to define shared directories, set permissions, and configure various options to secure our file shares.

Additionally, Samba supports integration with Windows workgroups and domains, allowing for a seamless file-sharing experience across different operating systems.

By using Samba, we can create a secure and efficient file-sharing environment that meets our needs or that of your organization. It provides the flexibility to manage file access and permissions while ensuring compatibility with various client systems.

### Virtual Private Network

A Virtual Private Network (VPN) is a technology that creates a secure and encrypted connection over a less secure network, such as the internet. It allows users to send and receive data as if their devices were directly connected to a private network, ensuring privacy and security. VPNs are widely used for secure remote access, protecting sensitive data, and maintaining anonymity online.

WireGuard is a modern VPN protocol designed to be simpler, faster, and more secure than traditional protocols like OpenVPN and IPSec. Developed by Jason Donenfeld, WireGuard uses state-of-the-art cryptography and is implemented with only about 4,000 lines of code, making it easier to audit and maintain. It employs the ChaCha20 encryption algorithm, which is known for its speed and security. WireGuard operates within the Linux kernel, enhancing performance and reducing latency. Its lightweight nature and efficiency make it an excellent choice for both local server security and remote management.

By integrating WireGuard on our server setup, we can ensure robust encryption and seamless remote access, providing a secure environment for managing our local server.

### Firewall

A firewall is a crucial component in network security, acting as a barrier between a trusted internal network and untrusted external networks. It monitors and controls incoming and outgoing network traffic based on predetermined security rules. Firewalls can be hardware-based, software-based, or a combination of both, and they help protect against unauthorized access, cyber-attacks, and data breaches.

UFW (Uncomplicated Firewall) is a user-friendly interface for managing iptables, the underlying firewall system in many Linux distributions. UFW simplifies the process of configuring a firewall, making it accessible even to those who are not experts in network security. It is included by default in Ubuntu and other Debian-based distributions.

Using UFW for firewall management in our project will help ensure that our local server is secure, while still allowing necessary remote management access.

### Implementing Fail2Ban for Enhanced Server Security

In the context of designing, installing, and configuring a secure local server with secure remote management, implementing robust security measures is paramount. One such measure is the use of Fail2Ban, an intrusion prevention software framework.

Fail2Ban is an open-source intrusion prevention software written in Python. It is designed to protect servers from brute-force attacks by monitoring log files for repeated failed login attempts and other suspicious activities. When such patterns are detected, Fail2Ban updates firewall rules to temporarily ban the offending IP addresses, thereby mitigating potential security threats.

The primary reason for using Fail2Ban is to enhance the security of a server by preventing unauthorized access. Brute-force attacks, where attackers try numerous combinations of usernames and passwords to gain access, are a common threat. Fail2Ban helps in reducing the risk of such attacks by automatically blocking IP addresses that exhibit malicious behavior. This not only protects sensitive data but also ensures the stability and reliability of server operations.

Fail2Ban is crucial for maintaining the security and integrity of a server. By automatically banning IP addresses that exhibit malicious behavior, it reduces the likelihood of successful brute-force attacks. This is especially important for servers that are accessible over the internet, as they are more susceptible to such attacks. Additionally, Fail2Ban can be configured to protect various services, making it a versatile tool in a server administrator's security arsenal.

### Monitoring

Nagios is a powerful open-source monitoring tool widely used for server and network monitoring. It helps ensure the availability and performance of your IT infrastructure by providing comprehensive monitoring capabilities. Nagios can monitor various server metrics, including CPU load, disk usage, and network traffic, and it supports both agent-based and agentless monitoring.

Using Nagios for server monitoring is essential for several reasons. It allows for early detection of issues, reducing downtime and improving overall system reliability. By continuously monitoring server health, Nagios can alert administrators to potential problems before they escalate, ensuring timely intervention. Additionally, Nagios offers extensive customization options, enabling you to tailor the monitoring setup to your specific needs.

To install and configure Nagios, you typically start by setting up Nagios Core on a central server. You can then install Nagios agents, such as NRPE, on remote servers to collect data. The collected data is displayed on a web interface, providing real-time insights into your server's performance. This setup is crucial for maintaining a secure and efficient server environment, especially when managing remote clients.

By integrating Nagios into your server management strategy, you can enhance the security and reliability of your local server while ensuring effective remote management.

### Data Backup

The **3-2-1 backup strategy** is a widely recommended approach for ensuring data protection and recovery. It involves maintaining three copies of your data: one primary copy and two backups. These backups should be stored on two different types of media, such as an internal hard drive and an external storage device, to enhance redundancy. Additionally, one of these copies should be kept off-site, away from your primary location, to safeguard against local disasters like fires or floods. This strategy helps mitigate risks, ensuring that even if one backup fails, you still have access to your critical data. By following the 3-2-1 rule, you can significantly enhance the security and reliability of your data management system.

Lastly, document all configurations and changes made to the server for maintenance and troubleshooting purposes.

This setup will provide a robust foundation for a secure local server with remote management capabilities, suitable for small to medium-sized enterprises or personal projects requiring secure access and file sharing capabilities. We should always remember to adhere to best practices for security and regularly review server's security posture to mitigate potential risks.

# IMPLEMENTATION

## IMPLEMENTATION STEPS

### Server Hardware and OS Installation

#### **Server Hardware**

We are installing the server on a desktop computer with the following specifications:

**Processor:**  Intel Dual Core 64bits with a processing speed of 3.0GHz.

**RAM:** 4GB.

**Storage:** 300GB+ Extensible.

**Network:** Network Interface Card for Wired LAN and a TP-LINK wireless USB Network Adapter for Wireless LAN.

#### **OS Installation**

First let’s download the latest ubuntu Server from its official website: https://ubuntu.com/download/server

* The latest version is ubuntu Server 24.4 LTS.
* *While ubuntu server is downloading, we need to download a disk imaging program which will be used to create a bootable CD-Drive or flash drive of the ubuntu server OS.*
* *For this purpose, we will download Rufus from its official website.*
* *When the OS download is finish, we plug in a USB flash drive of at least 8gb of free space and launch Rufus.*
* *In Rufus we select USB drive under the ‘Device’ section and choose ‘Disk or ISO Image’ under the ‘Boot Selection’ section. click on the ‘SELECT’ button to browse and select your downloaded Ubuntu Server ISO file. Ensure that the ‘partition scheme’ is set to GPT if your system uses UEFI or MBR if it’s an older system that uses BIOS. Once you’ve made sure all settings are correct, click ‘Start’ to begin creating the bootable drive.*
* *After Rufus completes the process, we have a bootable Ubuntu USB drive.*
* *To install Ubuntu, insert the USB drive into the target computer and boot from it. Press the key to enter the boot menu (commonly F12, F2, ESC, or DEL, depending on the motherboard manufacturer). For our target machine, we press ‘DEL’ then ‘F8’ to boot from our bootable drive.*
* *Once booted from the USB, we are greeted by the Ubuntu installer.*
* Once we see the GRUB boot menu, we pick the first option to Try or install Ubuntu server.
* Now we follow the prompts on screen to install Ubuntu. Leave the OpenSSH Server checked to have remote access to our machine once it’s setup.
* \*\*we make sure to uncheck “Set up this drive as LVM group” when asked about partitioning the drive. \*\* if we leave that checked, it will split our drive into multiple volumes, using only part of the disk, and we would need to manually resize later, which complicates the process.
* Following the instructions, we enter a username and a password we can remember, this is very crucial because we’ll need this password to manage and administer our server configurations.
* Our Ubuntu Server has been completely Installed.

### Network Configurations

Once the installation is complete, we need to assign a static IP address to our Server on the network. This ensures that we always know our servers IP address on our local network so that we can connect over SSH and begin configurations.

* \*\*It is recommended to set this up in our router’s DHCP settings instead of assigning a static IP in our server’s network configurations. \*\*
* This will prevent future issues in case we need to upgrade or change network, or if any devices happen to use the IP address that we would’ve set manually.
* To accomplish this task, first we need to see our server’s IP address which was assigned to it by our router’s DHCP server, using the command:

*> ifconfig*

* *\*\*If by any chance you run the above command and it fails, reporting command or package not found, make sure to run the following commands and try again. \*\**

*> sudo apt-get update \*\*[This command updates the server’s package libraries.]*

*> sudo apt install net-tools \*\*[This command will install the network tools package which contain the “ifconfig” command.]*

* Now we need to identify our IP address from out network interface. Since we are using a wireless connection, our network interface is “wlx9c53224b3e89” which is our wireless interface and our IP address is “192.168.1.124”.
* Now that we have our IP address, we go to our router’s web interface, go to advanced settings, then > router and > DHCP section. We find the option related to static leases, on our router, it is IP and MAC Address Binding. Find our servers name and MAC Address through the list of connected devices and set a Static IP.
* Once we apply the changes, we need to restart our router and server to get our network configurations set.

### Remote Access

#### **3.1 Configuration on Server**

Now that our static IP address it set, we’ll write a shell script to verify that we have the OpenSHH server protocol installed and running on our server.

*#!/bin/bash*

*# A script that checks if OpenSSH server is installed and runs the service if not*

*# Check if OpenSSH is installed*

*if ! ssh -V > /dev/null 2>&1; then*

*echo "OpenSSH is not installed."*

*read -p "Do you want to install it? (y/n): " install\_openssh*

*if [ "$install\_openssh" = "y" ]; then*

*sudo apt update && sudo apt install openssh-server*

*echo "OpenSSH server has been successfully installed!"*

*else*

*echo "Installation cancelled."*

*exit 1*

*fi*

*fi*

*# Check if OpenSSH is running*

*if ! sudo systemctl is-active --quiet ssh; then*

*echo "OpenSSH server is installed but not running."*

*read -p "Do you want to start the SSH service? (y/n): " run\_ssh*

*if [ "$run\_ssh" = "y" ]; then*

*sudo apt-get update*

*sudo apt-get install --only-upgrade openssh-server*

*if ! sudo systemctl start ssh; then*

*sudo apt install openssh-server*

*sudo systemctl start ssh*

*echo "SSH is up and running."*

*sudo systemctl status ssh*

*fi*

*else*

*echo "SSH service not started."*

*exit 1*

*fi*

*else*

*echo "OpenSSH is already installed and running."*

*fi*

Let's break down the script into its sections and lines:

1. \*\*Shebang Line (#!/bin/bash): \*\*

- The first line, starting with `#!/bin/bash`, is called the "shebang" or "hashbang." It specifies the interpreter to use for executing the script. In this case, it indicates that the script should be interpreted using the Bash shell.

2. \*\*Comments: \*\*

- Comments in Bash scripts begin with the `#` symbol. They provide explanations or context for the code.

- The comments in our script describe what the script does and guide the reader through its functionality.

3. \*\*Checking if OpenSSH is Installed: \*\*

- The script checks whether OpenSSH is installed by running the `ssh -V` command.

- If the command fails (returns a non-zero exit status), it means OpenSSH is not installed, and the script proceeds to prompt the user whether they want to install it.

4. \*\*User Interaction (Read and Conditional Statements): \*\*

- The `read` command allows the user to input a response.

- The script stores the user's response (either "y" or "n") in the `install\_openssh` variable.

- If the user chooses to install OpenSSH (`"$install\_openssh" = "y"`), the script updates the package list (`sudo apt update`) and installs the `openssh-server` package.

- If the user cancels the installation, the script exits with an exit code of 1.

5. \*\*Checking if OpenSSH is Running: \*\*

- The script uses `sudo systemctl is-active --quiet ssh` to check if the OpenSSH service is running.

- If the service is not active, it prompts the user whether they want to start it.

- Similar to the previous section, the user's response is stored in the `run\_ssh` variable.

- If the user chooses to start the SSH service (`"$run\_ssh" = "y"`), the script updates the package list again and starts the service.

- If the service fails to start, it tries to install the package and start the service again.

6. \*\*Final Check: \*\*

- If OpenSSH is already installed and running, the script informs the user.

Remember that this script is designed to automate the installation and management of the OpenSSH server.

We now have ssh up and running, to connect to our server, we should make sure our client is connected to the same network as our server.

#### **3.2 On a Windows client**

* Launch a terminal (command line or Powershell), check clients IP address by typing the command
* *> ipconfig*
* On our windows client, we are connected to a wireless interface:

*Wireless LAN adapter Wi-Fi:*

*Connection-specific DNS Suffix . : home*

*IPv6 Address. . . . . . . . . . . : fd02:e28:fcdf:8400:df10:f632:eb14:cc01*

*Temporary IPv6 Address. . . . . . : fd02:e28:fcdf:8400:70dc:b4a9:add6:7921*

*Link-local IPv6 Address . . . . . : fe80::31b1:397d:c2ae:3f20%10*

***IPv4 Address. . . . . . . . . . . : 192.168.1.104***

*Subnet Mask . . . . . . . . . . . : 255.255.255.0*

*Default Gateway . . . . . . . . . : 192.168.1.1*

We know our IP Address, we need to make sure OpenSHH client is installed and running on our client, by following these steps:

*Open “Settings” > “System” > “Optional Features”*

* *Click “View optional features”, search “OpenSSH Client” and select “OpenSSH Client”, then “Install” (only possible with admin rights)*
* *Open the “Services” app in Windows and set the startup type for “OpenSSH Authentication Agent” and “OpenSSH client” to “Manual”, making sure any remote connection is intentional and not made automatically*

Ssh is up and running on our client, next let’s connect to our server via it.

Launch terminal run the command:

* *ssh server-username@server\_ip*

example

* *ssh localhost@192.168.1.124*

Type in the password. Most password prompts for Linux utilities don’t show the password whiles typing, so just type and press enter. We are in!!!

Next step is to secure our ssh connection is a public key authentication.

##### **3.1.1 SSH Key Authentication**

We need to generate a ssh key pair for use in ssh, first we need to make sure that we don’t have a ssh key pair on our system already.

On our terminal connected to the server, type the command:

> *ls -l ~/.ssh*

if “total 0” then we don’t have a pair setup on the server therefore we need to generate it.

On our windows client, open another terminal preferably powershell, type in the command:

* *ssh-keygen -b 4096*
* -b lets us specify the bit size
* it will then prompt for us to specify where the generated key pair should be installed, press “Enter” to use the default location.
* Next, it will prompt for password, if we put a password, we need to make sure to remember it because if we loss it we can’t not recover it again, password adds an extra layer of security. We’ll just press “Enter” twice to not use a password. Key pair have been generated in the default location.
* Now that we have our key pair, we need to copy the public key to our server.
* Navigate to the local save point of the ssh key pair and copy the public key. C:\users\[yourusername]\.ssh\id\_rsa.pub for example:

*C:\users\localhost\.ssh\id\_rsa.pub*, this is where our generated public key is found. Open the file “id\_rsa.pub in notepad and copy the public key.

* Switch to the terminal connected to the server and create a file called “authorized\_key” in the .ssh folder found in the server’s home directory using the command:
* *vim ~/.ssh/authorized\_keys*
* paste in the public key, save and close the file.
* Close connection to the server by typing > *exit*. Retry connecting to the server. If the connection process is direct without asking password authentication, then, ssh authentication is successful.

#### **3.3 On a Linux Client**

On a Linux client, the steps are similar to those above with some small differences. Here to check the IP address, the command we use is:

* *ipconfig*

Identify the connected network interface and note the IP address.

Verify ssh is installed, up and running using the same script we used in the server environment.

Now let connect to our server: > [localhost@192.168.1.124](mailto:localhost@192.168.1.124), on the password prompt type in password and continue.

##### **3.3.1 SSH Key Authentication**

To generate the ssh key pair, open another terminal and run the command:

* ssh-keygen -b 4096
* it will then prompt for us to specify where the generated key pair should be installed, press “Enter” to use the default location.
* Next, it will prompt for password, if we put a password, we need to make sure to remember it because if we loss it we can’t not recover it again, password adds an extra layer of security. We’ll just press “Enter” twice to not use a password. Key pair have been generated in the default location.
* Now that we have our key pair, we need to copy the public key to our server.

Linux client has a command for this:

* ssh-copy-id [localhost@192.168.1.124](mailto:localhost@192.168.1.124)

Close the connection to the server and type in the command above in the terminal, it will copy the public key automatically to ~/.ssh/authorized\_keys in the server thus, automating the process. Now try connecting to the server, it will connect without asking password authentication.

After setting SSH Key Pair authentication for all clients, edit ssh configuration file in the server, find the line “# PasswordAuthentication yes” and modify by uncommenting the line and replace “yes” with “no”, remove the “#” symbol in front of the line.

* sudo vim /etc/ssh/ssh\_config

### File Sharing

The implementation of a Samba service consists of configuring a file sharing and printing server to allow clients, especially on Windows, to access shared resources on a network. This involves installing the Samba software, configuring settings, creating the appropriate shares and permissions, and managing users and groups.

We will write a script to automate the process on our server.

#### Configuration on Server

*#!/bin/bash*

*# Check if Samba is installed and run it*

*if ! smbd -V > /dev/null 2>&1; then*

*echo "Samba is not installed!"*

*read -p "Do you want to install Samba? (y/n): " install\_smbd*

*if [ "$install\_smbd" = "y" ]; then*

*sudo apt update && sudo apt install samba*

*echo "Samba installed successfully."*

*else*

*echo "Installation aborted."*

*exit 1*

*fi*

*fi*

*# Check if smbd is running*

*if ! sudo systemctl is-active --quiet smbd; then*

*echo "Samba is installed but not running."*

*read -p "Do you want to start the smbd service? (y/n): " run\_smbd*

*if [ "$run\_smbd" = "y" ]; then*

*sudo systemctl start smbd*

*sudo systemctl status smbd*

*else*

*echo "smbd service aborted."*

*exit 1*

*fi*

*else*

*echo "Samba is up and running!"*

*fi*

*# Create a user for Samba*

*echo "Let's create a user for Samba."*

*read -p "Enter a username (leave blank to use system $USER): " user\_name*

*if [ -z "$user\_name" ]; then*

*echo "Enter password for $USER on prompt."*

*sudo smbpasswd -a "$USER"*

*echo "$USER Samba user has been created successfully!"*

*echo "Now adding $USER to the sambashare group..."*

*sudo usermod -aG sambashare "$USER"*

*else*

*echo "Creating $user\_name user for Samba..."*

*sudo useradd -m $user\_name*

*echo "Enter password for $user\_name on prompt."*

*sudo smbpasswd -a $user\_name*

*echo "Adding $user\_name to sambashare group."*

*sudo usermod -aG sambashare $user\_name*

*echo "$user\_name added to sambashare group."*

*fi*

*# Samba configuration variables*

*echo "Enter the name for the Samba share (or leave blank to use default 'myshare'):"*

*read share\_name*

*if [ -z "$share\_name" ]; then*

*share\_name="myshare"*

*fi*

*# Ask user for the relative path to the directory*

*echo "Enter the relative path to the directory you want to share."*

*echo "To use the default directory '/media/myfiles', leave it blank and press Enter."*

*read relative\_path*

*if [ -z "$relative\_path" ]; then*

*sudo mkdir -p /media/myfiles*

*path="/media/myfiles"*

*else*

*sudo mkdir -p "$HOME/$relative\_path"*

*path="$HOME/$relative\_path"*

*fi*

*smb\_conf="/etc/samba/smb.conf"*

*cp "$smb\_conf" "/etc/samba/smb.conf1"*

*# Adding configurations to Samba configuration file*

*sudo sed -i 's/map to guest = bad user/map to guest = never/g' "$smb\_conf"*

*echo "[$share\_name]" | sudo tee -a "$smb\_conf"*

*echo "comment = SambaShare: $share\_name" | sudo tee -a "$smb\_conf"*

*echo "path = $path" | sudo tee -a "$smb\_conf"*

*echo "browseable = yes" | sudo tee -a "$smb\_conf"*

*echo "readonly = no" | sudo tee -a "$smb\_conf"*

*echo "guest ok = no" | sudo tee -a "$smb\_conf"*

*echo "valid users = $user\_name,$USER" | sudo tee -a "$smb\_conf"*

*# Restart server to load all changes*

*sudo systemctl restart smbd*

*echo "Samba share \"$user\_name,$USER\" has been created successfully for directory \"$path\"."*

Let's break down the script step by step:

1. \*\*Checking if Samba is Installed: \*\*

- The script checks whether Samba is installed by running the `smbd -V` command.

- If the command fails (returns a non-zero exit status), it means Samba is not installed, and the script proceeds to prompt the user whether they want to install it.

2. \*\*User Interaction (Read and Conditional Statements): \*\*

- The `read` command allows the user to input a response.

- If the user chooses to install Samba (`"$install\_smbd" = "y"`), the script updates the package list (`sudo apt update`) and installs the `samba` package.

- If the user cancels the installation, the script exits with an exit code of 1.

3. \*\*Checking if Samba is Running: \*\*

- The script uses `sudo systemctl is-active --quiet smbd` to check if the Samba service is running.

- If the service is not active, it prompts the user whether they want to start it.

- If the user chooses to start the Samba service (`"$run\_smbd" = "y"`), the script starts the service and displays its status.

4. \*\*Creating a User for Samba:\*\*

- The script prompts the user to enter a username (or use the system username if left blank).

- If a custom username is provided, the script creates a new user using `sudo useradd -m $user\_name`.

- In both cases, it sets the Samba password for the user using `sudo smbpasswd -a`.

5. \*\*Adding User to sambashare Group:\*\*

- The script adds the user to the `sambashare` group using `sudo usermod -aG sambashare`.

6. \*\*Samba Share Configuration:\*\*

- The script prompts the user to enter a name for the Samba share (default is "myshare").

- The script prompts the user to enter a relative path to the directory they want to share.

- If the user leaves it blank (presses Enter), the default directory `/media/myfiles` is created using `sudo mkdir -p /media/myfiles`.

- Otherwise, if the user provides a custom relative path, the script creates the directory under the user's home directory using `sudo mkdir -p "$HOME/$relative\_path"`.

7. \*\*Backup Samba Configuration File: \*\*

- The script makes a backup copy of the original Samba configuration file (`/etc/samba/smb.conf`) by copying it to `/etc/samba/smb.conf1`.

8. \*\*Modifying Samba Configuration: \*\*

- The `sudo sed -i 's/map to guest = bad user/map to guest = never/g' "$smb\_conf"` line replaces the `map to guest` setting in the Samba configuration file. It ensures that guests are never mapped to a user account.

- The subsequent lines add configurations for the Samba share:

- `[$share\_name]`: Defines the section for the share.

- `comment`: Provides a description for the share.

- `path`: Specifies the path to the shared directory.

- `browseable`: Allows the share to be visible when browsing network resources.

- `readonly`: Sets the share as read-only (no write access).

- `guest ok`: Disallows guest access.

- `valid users`: Lists the valid users (including the system user and the Samba user).

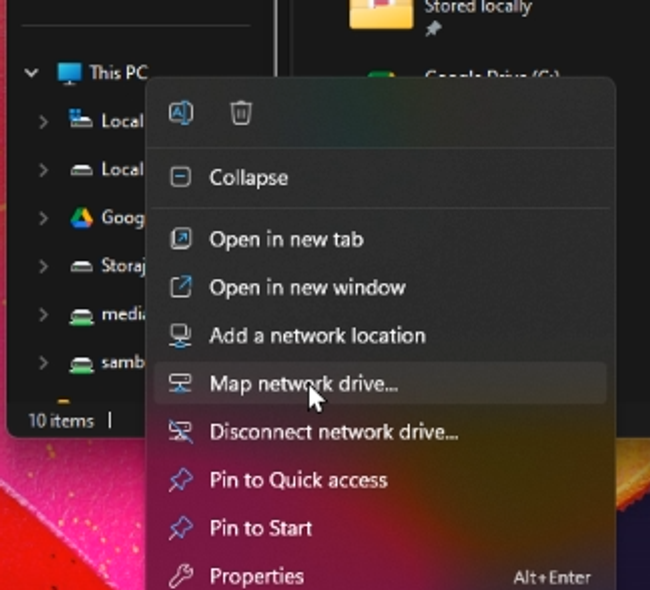
9. \*\*Restarting the Samba Service: \*\*

- The script restarts the Samba service using `sudo systemctl restart smbd` to apply the changes made to the configuration file.

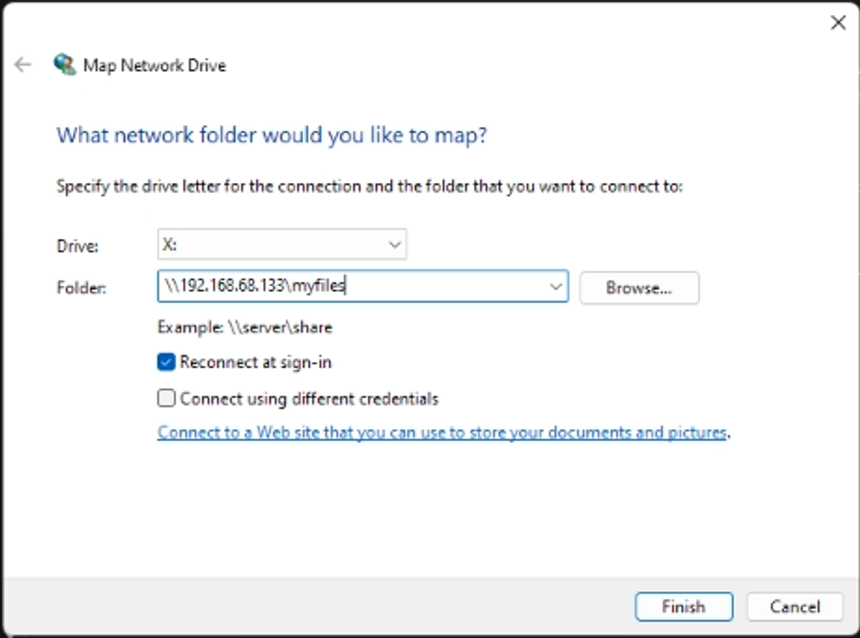
10. \*\*Final Message: \*\*

- The script displays a success message indicating that the Samba share has been created successfully for the specified directory.

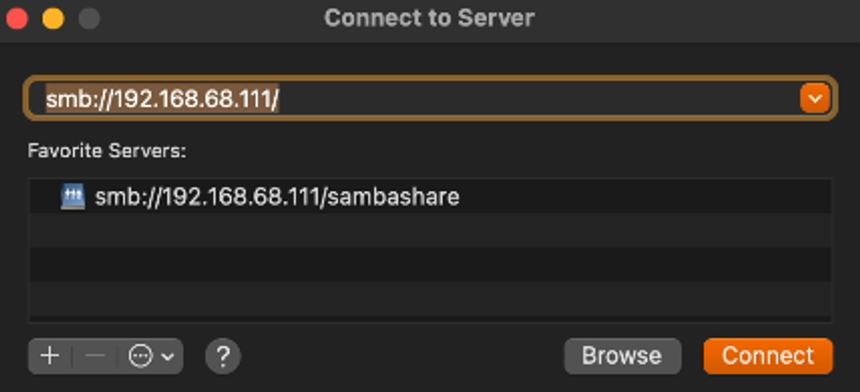
#### **4.2 Configuration on your Client - Windows, macOS**

On your Windows PC, you can right-click This PC in the Explorer, and select Map network drive.

Input two backslashes followed by the IP of your server, make sure it’s valid by clicking “Browse” and seeing if your files are in there as they should be.



On the Mac, you can connect to it by opening Finder, selecting Go from the drop-downs, and clicking Connect to Server, where you input smb:// followed by the IP of your server and click Connect.



### Virtual Private Network

#### **5.1 WireGuard**

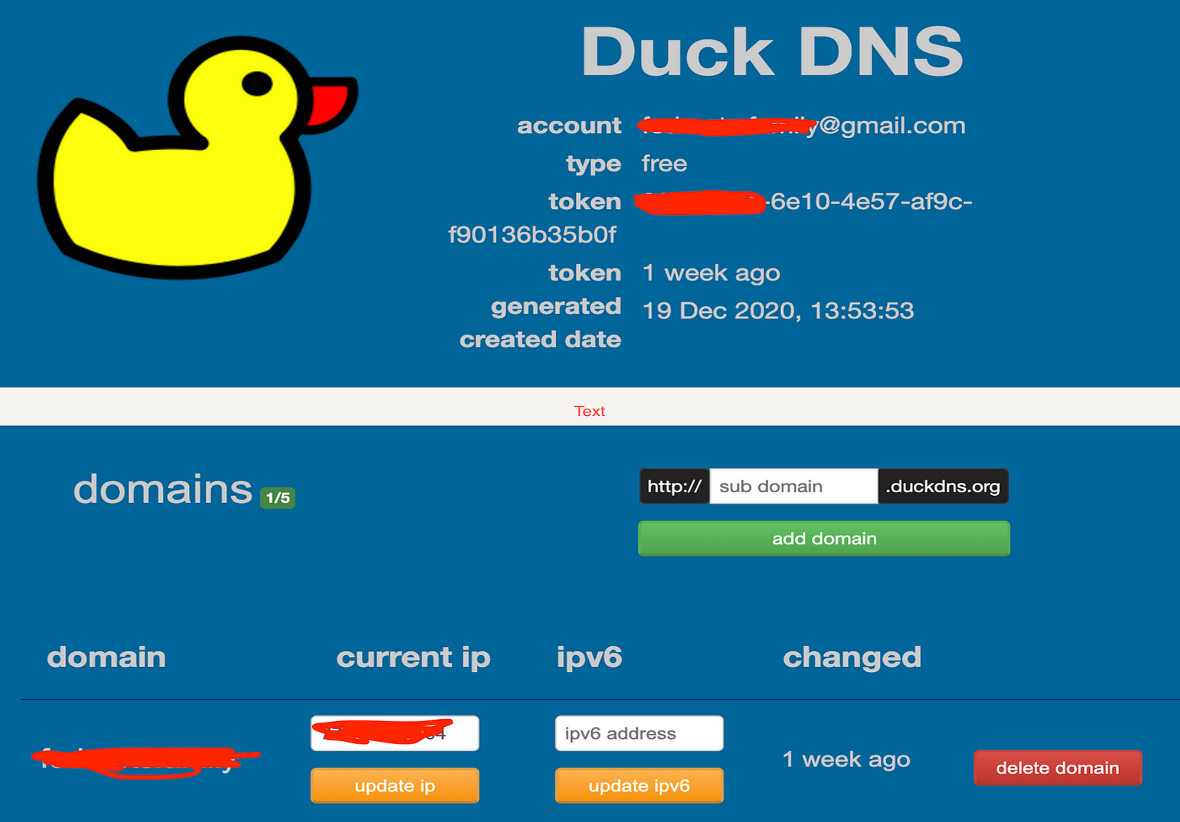
Wireguard is a free and open-source VPN. Wireguard can be used to access the devices on a home network from outside of the home network without having to port forward all the services. This greatly increases the network’s security, as less services are exposed and there are less entry points for the network.

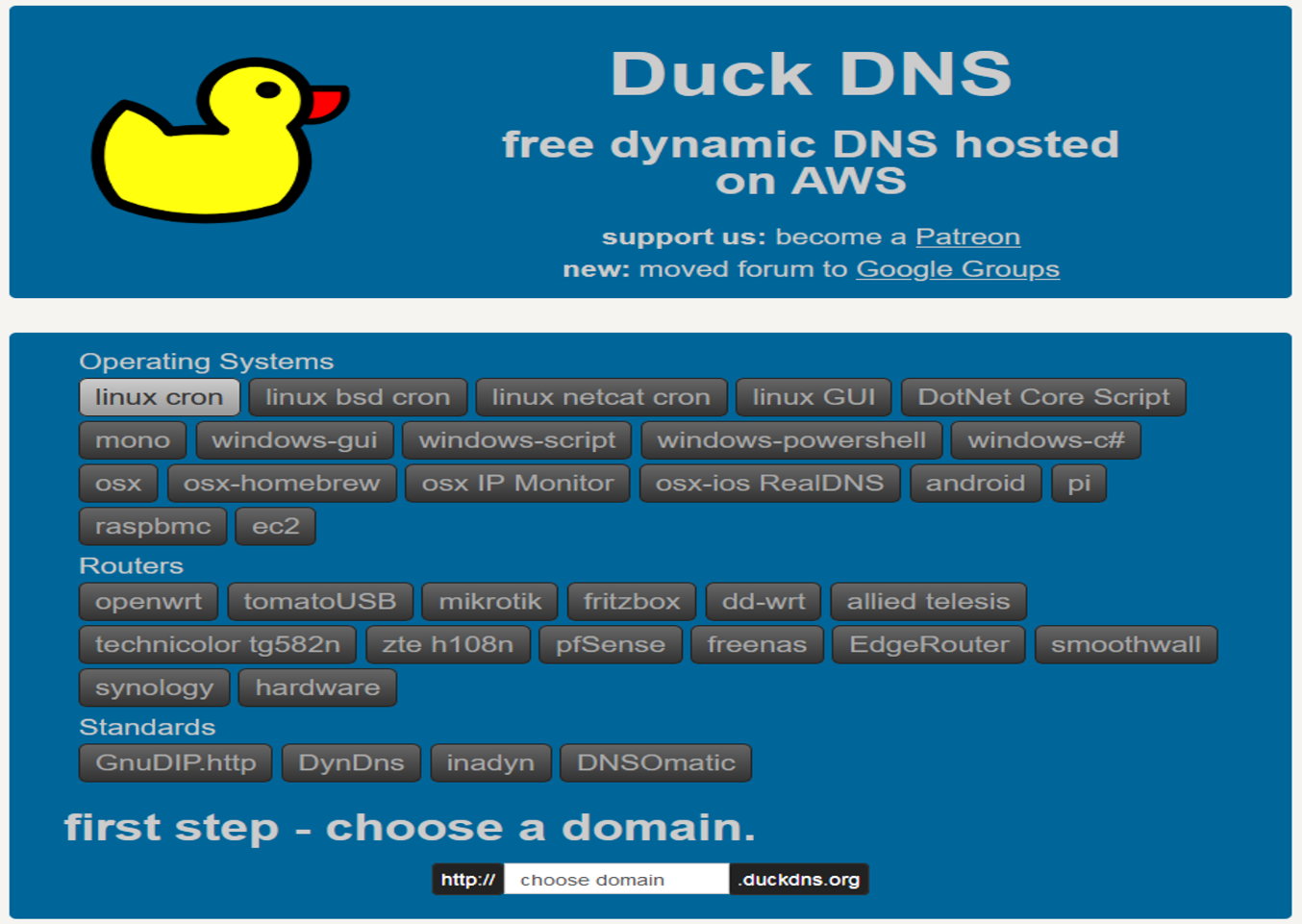
#### **5.1.1 Dynamic DNS Setup**

A DDNS is required if we need to find our home network and we don’t have a static IP address. Most ISPs don’t provide a static IP address or change your IP address periodically. “If you don’t know if you have a static IP address”, it is best to air on the side of caution and set up a DDNS regardless.

A DDNS is a service that will give us a hostname (a web address) that always points to our home network’s public IP address. There are many DDNS services, and any one of them would work, but we’ll use DuckDNS as it’s free, permanent and quite user friendly.

Go to [duckdns.org](http://duckdns.org) and sign in using your preferred method. Then, create a domain with any name that you like, as long as it isn’t already taken.



Then, click on install, go the Linux Cron section, and follow the instructions you see on the page.

#### **5.1.2 Port Forwarding**

In order for Wireguard to work, we need to forward port 51820 in our router’s web interface. This varies on each router, so you will have to find how to do it in your router specifically. On our router, we go to > Settings, Advanced, Security, then Special Applications. We make sure to select the IP of our server as the LAN IP Address, 51820 as the port on both WAN and LAN and use UDP as the protocol.

#### **5.1.3 Install**

To install Wireguard, we use the PiVPN scripts. Although they are designed to run on Raspberry Pi OS, they also work on Ubuntu Server, as they are both based on Debian.

Run > *curl -L [<https://install.pivpn.io>](<https://install.pivpn.io/>) | bash* and follow the prompts for Wireguard.

We don’t worry about the DHCP stuff in the installation, we’ve reserved your server’s local IP address in your router settings earlier. Leave the port as the default value, port 51820, which is the one we forwarded before. You can select any DNS provider you want; we’re going to scroll down to select CloudFlare. After this step is the important part – **we** **make sure to select “DNS Entry” instead of the IP address that shows up. Here, we’re going to input the DuckDNS address we made earlier.** We also recommend enabling unattended upgrades, for extra security.

##### **5.1.3.1 Setting up clients**

Once we’ve rebooted our machine after installation, we’re going to run *pivpn add* and give a name to our client. It can be anything you want, really.

Next, run *pivpn -qr.*

Now we go on our phone or whatever other device we want, and open the Wireguard app. Add the configuration by scanning the QR code on the screen.

From this point on, we can turn on the toggle switch that connects to our home network, and route all traffic through there. If you can see that both “Data Sent” and “Data Received” are going up, the connection works.

### 6. Firewall

#### 6.1 UFW (Uncomplicated Firewall)

We will write a script will set default settings for incoming and outgoing connections, we will allow ssh connections for specific IP addresses on port 22, allow WireGuard, samba and https.

*#!/bin/bash*

*#check if ufw is installed, if it is not, then install*

*if ! ufw --version > /dev/null 2>&1; then*

*read -p "ufw is not installed, Do you want to install? (y/n)" ufw\_install*

*if "$ufw\_install" = "y"; then*

*echo "Updating System library"*

*sudo apt update*

*echo "Installing UFW"*

*sudo apt install ufw*

*else*

*echo "Installation Aborted"*

*fi*

*else*

*echo "UFW is already Installed"*

*fi*

*#Check if ufw is enabled, if enabled then Disenable to configure*

*ufw\_status=$(sudo ufw status | grep -qw active; echo $?)*

*if [[ $ufw\_status -eq 0 ]]; then*

*echo "UFW is enabled"*

*echo "Now Disabling...."*

*sudo ufw disable*

*else*

*echo "UFW is not enabled..."*

*fi*

*#Configuring UFW rules*

*echo "Setting default policy for outgoing connections"*

*sudo ufw default allow outgoing*

*echo "Setting default policy for incoming connections"*

*sudo ufw default deny incoming*

*echo "Setting Policy for https"*

*sudo ufw allow 443*

*echo "Setting policy for WireGuard"*

*sudo ufw allow 51820/udp*

*echo "Setting policy for incomming ssh and samba connection"*

*sudo ufw allow 137/udp*

*sudo ufw allow 138/udp*

*sudo ufw allow 139/tcp*

*sudo ufw allow 445/tcp*

*read -p "Enter dynamic domain name you created" domain*

*if [ -z "$domain" ]; then*

*current\_ip=$(host "$domain" | awk '{print $NF}')*

*sudo ufw allow from "$current\_ip" to any port 22 proto tcp*

*echo "Enter Ip addresses (one per line, press ctrl+D when done):"*

*while read -r ip; do*

*sudo ufw allow from "$ip" to any port 22 proto tcp*

*sudo ufw allow from "$ip" to any app samba*

*done*

*# Reload and enable UFW*

*sudo ufw reload > /dev/null*

*sudo ufw enable*

*sudo ufw status verbose*

 Let’s break down the script line by line, explaining what each part does:

1. **Checking UFW Installation:**
   * *if ! ufw --version > /dev/null 2>&1; then*
     + This line checks if UFW is installed by running the ufw --version command.
     + If the command fails (returns a non-zero exit status), it means UFW is not installed.
     + The *> /dev/null 2>&1* part redirects any output (both standard output and error output) to /dev/null, effectively suppressing it.
2. **Prompting User for Installation:**
   * *read -p "ufw is not installed, Do you want to install? (y/n)" ufw\_install*
     + This line prompts the user with the message and stores their response (either ‘y’ or ‘n’) in the ufw\_install variable.
3. **Installing UFW (if requested):**
   * *if "$ufw\_install" = "y"; then*
     + If the user chose to install UFW (by entering ‘y’), the script proceeds.
   * *sudo apt update*
     + Updates the system package list.
   * *sudo apt install ufw*
     + Installs UFW.
4. **Disabling UFW (if enabled):**
   * *ufw\_status=$(sudo ufw status | grep -qw active; echo $?)*
     + Checks if UFW is currently enabled by running sudo ufw status and looking for the word “active.”
     + The result (0 or 1) is stored in the ufw\_status variable.
   * *if [[ $ufw\_status -eq 0 ]];* then
     + If UFW is enabled (status code 0), the script proceeds.
   * *sudo ufw disable*
     + Disables UFW.
5. **Configuring UFW Rules:**
   * Sets default policies:
     + *sudo ufw default allow outgoing*
     + *sudo ufw default deny incoming*
   * Specific rules are added:
     + sudo ufw allow 443 (HTTPS)
     + *sudo ufw allow 51820/udp* (WireGuard)
     + *sudo ufw allow 137/udp, sudo ufw allow 138/udp, sudo ufw allow 139/tcp, sudo ufw allow 445/tcp* (Samba and other services)
   * Prompts user for a dynamic domain name:
     + *read -p "Enter dynamic domain name you created" domain*
     + If provided, allows incoming SSH traffic from that domain’s IP address.
   * Allows SSH and Samba traffic from specified IP addresses (entered by the user).
6. **Reloading and Enabling UFW:**
   * *sudo ufw reload > /dev/null*
     + Reloads UFW rules.
   * *sudo ufw enable*
     + Enables UFW.
   * *sudo ufw status verbose*
     + Displays the verbose status of UFW rules.

### Implementing Fail2Ban for Enhanced Server Security

Implementing Fail2Ban involves several steps:

1. \*\*Installation\*\*: Fail2Ban can be installed using package managers like `apt` for Debian-based systems or `yum` for Red Hat-based systems. For example, on a Debian-based system, you can install it using the command `sudo apt-get install fail2ban`.

2. \*\*Configuration\*\*: After installation, Fail2Ban needs to be configured to monitor specific log files and define the actions to be taken when suspicious activities are detected. This is done by editing the configuration files located in `/etc/fail2ban/`.

3. \*\*Creating Jails\*\*: A "jail" in Fail2Ban is a combination of a filter and an action. Filters define the patterns to look for in log files, while actions specify what to do when those patterns are found. Common jails include those for SSH, Apache, and other services.

4. \*\*Starting the Service\*\*: Once configured, the Fail2Ban service needs to be started and enabled to run at boot. This can be done using systemctl commands like `sudo systemctl start fail2ban` and `sudo systemctl enable fail2ban`.

### Monitoring

Here’s a detailed guide on implementing and configuring Nagios for monitoring our secure local server:

*#!/bin/bash*

*# Variables*

*read -p "Enter Nagios Version to install" NAGIOS\_VERSION*

*read -p "Enter User for nagios or leave blanc to use default 'NAGIOS\_USER=nagios'" NAGIOS\_USER*

*if [[ ! -z $NAGIOS\_USER ]]; then*

*NAGIOS\_USER="nagios"*

*fi*

*read -p "Enter Group for nagios or leave blanc to use default 'NAGIOS\_GROUP=nagcmd" NAGIOS\_GROUP*

*if [[ ! -z $NAGIOS\_GROUP ]]; then*

*NAGIOS\_GROUP="nagcmd"*

*fi*

*read -p "Enter Admin name for nagios or leave blanc to use default 'NAGIOS\_ADMIN=nagiosadmin" NAGIOS\_ADMIN*

*if [[ ! -z $NAGIOS\_ADMIN ]]; then*

*NAGIOS\_ADMIN="nagiosadmin"*

*fi*

*read -p "Enter your server ip" SERVER\_IP*

*# Install dependencies*

*sudo apt update*

*sudo apt install -y autoconf bc gawk dc build-essential gcc libc6 make wget unzip apache2 php libapache2-mod-php libgd-dev libmcrypt-dev make libssl-dev snmp libnet-snmp-perl gettext*

*# Download and extract Nagios*

*cd /tmp || exit*

*wget https://github.com/NagiosEnterprises/nagioscore/releases/download/nagios-"$NAGIOS\_VERSION"/nagios-"$NAGIOS\_VERSION".tar.gz*

*tar -xzf nagios-"$NAGIOS\_VERSION".tar.gz*

*cd nagios-"$NAGIOS\_VERSION" || { echo "Failed to change directory to nagios-$NAGIOS\_VERSION"; exit 1; }*

*# Configure and install Nagios*

*./configure --with-httpd-conf=/etc/apache2/sites-enabled*

*make all*

*sudo make install-groups-users*

*sudo usermod -aG "$NAGIOS\_GROUP" www-data*

*sudo make install*

*sudo make install-daemoninit*

*sudo make install-commandmode*

*sudo make install-config*

*sudo make install-webconf*

*# Enable CGI and restart Apache*

*sudo a2enmod rewrite cgi*

*sudo systemctl restart apache2*

*# Create Nagios admin user*

*sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users "$NAGIOS\_ADMIN"*

*# Configure firewall*

*sudo ufw allow apache*

*sudo ufw enable*

*sudo ufw reload*

*# Install Nagios plugins*

*sudo apt install -y monitoring-plugins nagios-nrpe-plugin*

*# Create configuration directory*

*sudo mkdir -p /usr/local/nagios/etc/servers*

*# Edit Nagios configuration files*

*sudo sed -i "s/^cfg\_dir=.\*$/cfg\_dir=\/usr\/local\/nagios\/etc\/servers/" /usr/local/nagios/etc/nagios.cfg*

*sudo sed -i "s/^$USER1$=.\*$/$USER1$=\/usr\/lib\/nagios\/plugins/" /usr/local/nagios/etc/resource.cfg*

*# Define NRPE command*

*echo 'define command{' | sudo tee -a /usr/local/nagios/etc/objects/commands.cfg*

*echo '    command\_name check\_nrpe' | sudo tee -a /usr/local/nagios/etc/objects/commands.cfg*

*echo '    command\_line $USER1$/check\_nrpe -H $HOSTADDRESS$ -c $ARG1$' | sudo tee -a /usr/local/nagios/etc/objects/commands.cfg*

*echo '}' | sudo tee -a /usr/local/nagios/etc/objects/commands.cfg*

*# Start and enable Nagios service*

*sudo systemctl start nagios*

*sudo systemctl enable nagios*

*sudo systemctl status nagios*

*# Restart Apache*

*sudo systemctl restart apache2*

*echo "Nagios installation and configuration completed!"*

*echo "Open Web Browser:*

*Navigate to http://$SERVER\_IP/nagios"*

*echo "Login with username: $NAGIOS\_ADMIN and password: the password you set earlier!"*

 Let’s break down the script and explain each part:

1. **User Input:**
   * The script prompts the user to input various parameters:
     + NAGIOS\_VERSION: The desired Nagios version.
     + NAGIOS\_USER: The Nagios user (default is “nagios”).
     + NAGIOS\_GROUP: The Nagios group (default is “nagcmd”).
     + NAGIOS\_ADMIN: The Nagios admin username (default is “nagiosadmin”).
     + SERVER\_IP: The server’s IP address.
2. **Install Dependencies:**
   * Updates the package list and installs necessary dependencies.
3. **Download and Extract Nagios:**
   * Downloads the specified Nagios version and extracts it.
4. **Configure and Install Nagios:**
   * Configures Nagios with the specified Apache configuration.
   * Compiles and installs Nagios binaries and scripts.
   * Sets up user groups and permissions.
5. **Enable CGI and Restart Apache:**
   * Enables CGI for Nagios web interface.
   * Restarts Apache to apply changes.
6. **Create Nagios Admin User:**
   * Creates an admin user for Nagios using htpasswd.
7. **Configure Firewall:**
   * Allows Apache through the firewall.
   * Enables and reloads the firewall rules.
8. **Install Nagios Plugins:**
   * Installs Nagios plugins.
9. **Create Configuration Directory:**
   * Creates a directory for server-specific configuration files.
10. **Edit Nagios Configuration Files:**
    * Updates nagios.cfg to include the server-specific configuration directory.
    * Sets the path for Nagios plugins in resource.cfg.
11. **Define NRPE Command:**
    * Adds a custom command for NRPE checks in commands.cfg.
12. **Start and Enable Nagios Service:**
    * Starts Nagios service.
    * Enables Nagios to start on boot.
    * Checks the status of Nagios service.
13. **Restart Apache:**
    * Restarts Apache to apply all changes.
14. **User Instructions:**
    * Informs the user that the installation and configuration are completed.
    * Provides instructions to open a web browser and log in to Nagios using the specified admin username and password.

### Data Backup

Implementing the **3-2-1 backup strategy** ensures data redundancy, resilience, and protection against data loss. Let’s break down how to implement it step by step:

1. **Three Copies of Your Data:**
   * **Primary Copy (On-Site):** Keep the original data on your secure local server.
   * **Secondary Copy (On-Site):** Create a backup copy on a different storage device within your premises (e.g., external hard drive, network-attached storage).
   * **Tertiary Copy (Off-Site):** Store another backup copy off-site (outside your physical location). This can be in the cloud or at a remote data center.
2. **Two Different Storage Types:**
   * **Local Storage (On-Site):** Use local storage media (e.g., external drives, NAS) for one of the backup copies.
   * **Cloud Storage (Off-Site):** Utilize cloud storage services (e.g., Amazon S3, Google Drive, Microsoft Azure) for the off-site backup. Cloud storage provides scalability, redundancy, and accessibility.
3. **One Off-Site Copy:**
   * Ensure that at least one backup copy is stored off-site. This protects against local disasters (fire, flood, theft) that could impact your primary data center.
   * Regularly verify the integrity of your off-site backups and periodically test data restoration.

Remember these best practices:

* **Automate Backups:** Schedule regular backups using backup software or scripts.
* **Encryption:** Encrypt your backups to protect sensitive data.
* **Versioning:** Maintain multiple versions of backups to recover from accidental deletions or corruption.
* **Test Restores:** Periodically restore data from backups to ensure they are functional.

### Troubleshooting

Document all configurations and changes made to the server for maintenance and troubleshooting purposes.

In conclusion, designing, installing, and configuring a secure local server with a secure remote management client is a critical task that ensures the integrity and accessibility of an organization's data and resources. By implementing robust security measures, such as encryption, firewalls, and regular updates, we can protect the server from potential threats and unauthorized access. Additionally, configuring a secure remote management client allows for efficient and safe management of the server from remote locations, enhancing operational flexibility and responsiveness. This comprehensive approach not only safeguards sensitive information but also supports the seamless operation of business processes, ultimately contributing to the overall success and resilience of the organization.

# CONCLUSION