ChatGPT

Confidential Documentation Vault

> ⚠ **Confidentiality Notice:** All content in this vault is **confidential and proprietary**. Unauthorized distribution, reproduction, or disclosure is strictly prohibited and may result in legal action.

Enterprise Infrastructure Blueprint

This document provides a complete deployment blueprint for our enterprise clients, detailing the end-to-end infrastructure design. It covers everything from physical layout to network architecture, ensuring a robust, scalable, and secure setup. The blueprint includes detailed network diagrams and hardware specifications to guide installations in a **server room or data center environment**.

An example of a secure server room setup with dedicated racks, cooling, and access control. The Enterprise Infrastructure Blueprint outlines how to achieve such a professional, organized environment.

Overview: Our enterprise infrastructure is designed with **privacy, reliability, and performance** in mind. Key elements of the blueprint include segregated network zones (e.g. **DMZ, internal LAN, and guest network**), high-availability server clusters, and on-site data storage using TrueNAS SCALE. All critical systems are on-premises to keep sensitive data under client control (no reliance on external cloud for core services). The blueprint also emphasizes redundant power and cooling, robust firewalling, and secure remote access via VPN (see WireGuard VPN section).

Physical Layout & Environment: We specify requirements for a dedicated, climate-controlled server room. For example:

- **Secure Location:** A lockable server room with solid-core or metal door (optional biometric access), surveillance cameras, and a fireproof safe for backups and sensitive materials.
- **Power & Cooling:** Redundant 20A power circuits with UPS backup and generator integration for power outages. Independent cooling (AC or ventilation) to maintain optimal temperature for equipment.
- **Rack and Cabling:** Standard 19-inch equipment racks with proper cable management. All servers, storage arrays, and network gear are rack-mounted. Cables are labeled and routed through patch panels for a clean layout. This minimizes signal interference and simplifies maintenance.
- **Environmental Monitors:** Temperature and humidity sensors, smoke detectors, and automated alerts for any environmental anomalies.

Network Architecture: The enterprise network diagram (provided in this document) illustrates a multi-tier architecture:

- **Edge Firewall/Router:** A high-performance firewall (e.g. pfSense or Cisco NGFW) at the network perimeter. It connects to the ISP and manages traffic into a **Demilitarized Zone (DMZ)** for any

public-facing services. The firewall implements a **default deny** stance, only allowing whitelisted traffic.

- **DMZ Segment:** Hosts external services (if any, such as a web server or a VPN endpoint) isolated from the internal network. This adds an extra layer of security—external clients can reach the DMZ servers but have no direct path to the internal LAN.
- **Internal Network Segments:** The internal LAN is segmented into VLANs or subnets by function (e.g. corporate PCs, IoT devices, servers, IP cameras). For enterprise clients, we recommend VLANs like *Workstations*, *Servers*, *VoIP/IoT*, and *Guest Wi-Fi*. Internal switches (Layer 2/3) handle VLAN tagging and inter-VLAN routing is tightly controlled (only allow necessary traffic between segments).
- **Core Services:** Core infrastructure like directory services (AD/LDAP), DNS/DHCP, and monitoring systems reside on the secure server VLAN. They are only accessible to authorized segments. Storage systems (NAS/SAN) connect to a storage network for high throughput and are also isolated from client subnets except via specified file-sharing protocols.
- **Remote Access:** No direct exposure of internal services to the internet. All remote access for administrators or clients is tunneled through a VPN (see WireGuard section) or other secure gateway in the DMZ. This ensures that even maintenance is done through encrypted channels with strict authentication.

Server & Storage Design: The blueprint calls for at least two server-grade machines for redundancy and load distribution:

- **Primary Server Node:** A powerful rackmount server (multi-core CPUs, ample RAM) running a hypervisor (Proxmox VE) to host critical VMs and containers. This primary node might run services like virtualization for business apps or AI workloads (see On-Prem AI Deployment Scripts), and potentially house an instance of our password vault or management VMs.
- **Secondary Server Node:** Another server (or high-end workstation) to run auxiliary services and provide failover. It might handle tasks like home automation controllers, media servers, or act as a hot spare for critical VMs (replicated from the primary).
- **GPU Acceleration:** If AI or heavy computation is required (as in many enterprise scenarios), one server is equipped with an **NVIDIA GPU** (configured via GPU passthrough on Proxmox). This allows VMs to leverage the GPU for machine learning inference, data analysis, or any GPU-intensive applications internally.
- **Storage Array (NAS):** Enterprise-grade storage is provided by TrueNAS SCALE on a dedicated NAS server or as a VM with direct disk access. The design uses ZFS with RAID for data protection (e.g. RAID-Z2 or mirrored vdevs with hot spares). At least 60–100TB raw capacity is recommended for heavy data users, using enterprise HDDs and SSD caches for performance. All sensitive data stays on this NAS; cloud backup is optional and encrypted if used. Local backup drives (encrypted external disks) are rotated off-site periodically and kept in the fireproof safe.

High-Level Diagram: The included network diagram visualizes the above components. It shows the internet feeding into a firewall, which then connects to a switch that feeds multiple VLANs (with example devices on each). Servers and the NAS are on the secure server VLAN. A management station (or laptop) can access the systems via an out-of-band management network or the VPN. The diagram also highlights how the WireGuard VPN server in the DMZ grants remote admins access to the management VLAN only, not the entire network.

Deliverables: Using this blueprint, our team will deliver to the client:

- A **professional network diagram** (physical and logical) tailored to their environment, printed in a binder and as a high-res PDF.
- A write-up of infrastructure specifications and **standard operating procedures (SOPs)** for routine tasks (e.g. how to start/stop services, replace a drive, etc.).
- Configuration files and credentials (securely stored, with backups in sealed envelopes in the safe).
- A training session and documentation for the client's IT staff (or family/assistants in a residential case) to understand the setup.

By following the Enterprise Infrastructure Blueprint, we ensure that every deployment is consistent with our high standards of security and reliability, providing a solid foundation for all other services (AI, storage, remote access) to run smoothly.

Advanced Security Protocols (Zero-Trust Architecture)

This document is a comprehensive guide to our **advanced security protocols**, built on a Zero-Trust architecture model. It details how we implement a "**never trust, always verify**" approach across the entire infrastructure, including network segmentation, firewall rules, authentication policies, and monitoring. By following these protocols, we greatly reduce the risk of breaches and ensure client data remains private and secure.

Strict physical security complements our digital security. The vault door and secure safe shown above are examples of how we protect critical hardware and backups from unauthorized access, aligning with our advanced security protocols.

Zero-Trust Principles: In a zero-trust model, no user or device is inherently trusted, even if it's inside the network. Every access request must be authenticated, authorized, and encrypted. We apply this principle in multiple layers:

- **Identity Verification:** Strong identity and access management (IAM) is mandatory. All user accounts (both for IT admins and end-users) use **multi-factor authentication (MFA)** wherever possible. For example, VPN access via WireGuard requires an SSH key or one-time pass along with the key, and administrative logins to servers require MFA or hardware tokens.
- **Least Privilege Access:** Each user and service is given the minimum privileges needed to perform its function. Admin accounts are separate from regular user accounts. Role-based access control (RBAC) is used on systems like TrueNAS and Proxmox – e.g., a "view-only" role for auditors vs. a "maintainer" role for system engineers. Credentials (passwords, keys) are stored in an encrypted vault (our on-prem password manager) and access to them is logged and limited.

Network Segmentation & Firewalling: Our network is segmented (as described in the Blueprint) and the firewall acts as a gatekeeper between all segments:

- **Firewall Rules:** The firewall configuration is provided as part of this guide. It operates on a **default deny** policy: all inbound **and** internal inter-VLAN traffic is denied unless explicitly allowed. We provide a set of predefined rules, such as: *allow* DNS from VLANs to the DNS server, *allow* web traffic

from workstations to the internet via proxy, *allow* IoT devices to reach only specific cloud endpoints, etc. Everything else (e.g., IoT to corporate network, or inbound from internet to LAN) is blocked.

- **Micro-Segmentation:** Even within a VLAN, critical services are further protected. For instance, the NAS shares may have host-based allow-lists (only known IPs can connect), and management interfaces (like the Proxmox web UI or TrueNAS UI) are restricted to a small management subnet or accessible only through VPN. If an attacker somehow lands on one internal machine, they shouldn't freely roam the network.
- **DMZ Isolation:** Any service in the DMZ (like the VPN endpoint or a web server if deployed) is on its own subnet with very limited access back to internal resources. The firewall may allow the DMZ service to query an internal authentication server (like LDAP) but never to initiate arbitrary connections into the LAN. This way, if the DMZ service is compromised, the damage is contained.

Secure Configuration & Protocols: We enforce modern, secure configurations on all systems:

- **Encryption Everywhere:** All data in transit is encrypted using strong protocols (HTTPS/TLS1.3 for web interfaces, WireGuard for VPN, SSH with key authentication for server access, etc.). Within the network, wherever possible, we prefer end-to-end encryption too – for example, if a client application talks to the NAS, it can use NFSv4 with Kerberos or SMB with AES encryption enabled.
- **Secure Defaults:** We disable legacy insecure protocols (e.g., SMBv1, Telnet, older SSL versions) and use hardened configurations (strong ciphers, certificates) across services. Administrative interfaces are not exposed on default ports publicly. We also enable enterprise features like **full-disk encryption** on servers and NAS (so if drives are stolen, data remains inaccessible).
- **Patching and Updates:** A protocol is in place for regular updates of all systems. The guide includes a maintenance schedule recommending at least monthly patching of OS and software (including firewall firmware, hypervisor updates, and application patches). Automatic security updates are enabled where possible, with notifications to admins. Each patch cycle is logged.

Monitoring & Response: Advanced security isn't complete without monitoring and incident response:

- **Logging:** All servers, network devices, and applications send logs to a central log server (or SIEM) where they are stored securely and analyzed for anomalies. For example, the firewall logs all blocked traffic and the VPN logs all connection attempts. Proxmox and TrueNAS logs are collected for any unusual activity (like repeated login failures).
- **Intrusion Detection/Prevention:** We deploy an IDS/IPS (such as **Suricata or Snort**) on the network, typically integrated with the firewall. This system uses updated threat signatures and behavioral analysis to alert or block suspicious traffic (like port scans or known exploit patterns). In our zero-trust setup, even internal traffic can be monitored for lateral movement indications.
- **Alerts:** The guide defines alert rules and contacts. If certain events occur (admin login outside business hours, a new device connecting to the network, etc.), the system sends immediate alerts (email/SMS) to the on-call security engineer. We also set thresholds (e.g., CPU spikes that might indicate crypto-mining malware) to trigger warnings.
- **Incident Response Plan:** Finally, we include a high-level incident response procedure. This covers isolating affected systems, collecting forensic data (from our logs and versioned backups), eradicating threats, and restoring from backups if needed. Because of the layered protections, a breach in one area is contained, but we assume breach and act swiftly whenever any anomaly is detected.

By rigorously following this **Advanced Security Protocols** guide, our deployments maintain a hardened posture. Even high-net-worth residential installs or small enterprise offices are secured to standards often seen in large corporate or government settings. Zero-trust ensures that **privacy is built-in by design** – only explicitly authorized activities are permitted, and continuous verification is the norm.

Client Contract Templates

This document contains our **legal agreement templates** used when engaging with clients. It's essentially a bundle of contract templates including a Master Services Agreement, Non-Disclosure Agreement, Service Level Agreement, and other key documents. Each template is written in clear legal language and can be customized per client/project. This section explains the purpose of each template and how to use them.

Overview: Maintaining confidentiality and clearly defining service terms is crucial in our line of business (ultra-secure, private technology solutions). The contract templates ensure that both our company and the client understand their obligations, protecting both parties. All templates are to be treated as confidential and should be provided to clients only after proper approvals (and typically after customizing with client-specific details).

Included Templates:

- **Master Services Agreement (MSA):** This is the core contract governing the overall relationship with the client. It outlines the general terms and conditions of our services. Key sections include scope of work, payment terms, warranty and liability clauses, confidentiality requirements, and termination conditions. The MSA is designed to be evergreen, covering any projects or services we perform for the client over time under one umbrella agreement.
- **Non-Disclosure Agreement (NDA):** A mutual NDA to protect sensitive information exchanged between us and the client. It ensures that any confidential data (e.g., network details, personal data, trade secrets) remains secret and is only used for the intended purposes. Our NDA template typically has no expiration for certain especially sensitive information and includes clauses about handling of data (e.g., requiring secure storage, return or destruction of info upon request). This NDA can be executed standalone or incorporated into the MSA.
- **Service Level Agreement (SLA):** This document defines the performance and support commitments. For managed services or ongoing support contracts, the SLA outlines things like uptime guarantees (e.g., 99.9% network uptime), support response times (e.g., critical issues addressed within 1 hour), maintenance windows, and penalties or credits if we fail to meet the service levels. Our SLA template is often an attachment to the MSA when we provide services like monitoring or regular maintenance.
- **Statement of Work (SOW):** For each specific project or installation, we have an SOW template. The SOW details the exact deliverables, milestones, timeline, and pricing for that project. It references the MSA for general terms but provides the specifics for that engagement (e.g., "Install a Tier-2 Private Digital Estate package at Client's residence including XYZ features, to be completed by Date"). We include acceptance criteria in SOWs so both parties agree on what "done" means.
- **Data Protection Addendum (DPA):** If the client's data includes personal or sensitive data (almost always the case), we include a DPA to address GDPR or other privacy law requirements. This template outlines how we handle and protect personal data as a service provider, our breach

notification duty, data retention, etc. It ensures compliance with privacy laws and typically is attached to the MSA when needed.

- **Other Miscellaneous Templates:** We also maintain templates for **Change Orders** (to document modifications to scope post-signature), **Acceptance Forms** (client sign-off that work is completed), and if applicable, **Equipment Lease or Finance Agreements** (in cases where we lease hardware to the client rather than sell outright). These are included in this vault for completeness but used as needed.

Usage Guidance: Each template is annotated with comments on how to fill in client-specific information (e.g., a cover sheet or highlighted sections to replace with names, dates, fees, etc.). When preparing a contract for a new client, follow these steps:

1. **Review and Customize:** Start with the MSA and NDA for all new clients. Fill in the client's name, address, the effective date, and tailor any service descriptions in exhibits. Remove any sections that don't apply and add specifics relevant to the client (for example, if the client has requested a unique provision or if local law requires an adjustment). Do the same for the SLA if we are committing to ongoing support or uptime, ensuring the service levels match what was promised in proposals.
2. **Add Project SOW:** Draft the Statement of Work for the initial project. Use the SOW template and include all details of the deployment (from the Blueprint and related docs). For instance, list the exact equipment to be installed, the security measures, and any training or support included. Be detailed—this prevents misunderstandings later. The SOW should reference the MSA as governing terms.
3. **Legal Review:** Although these templates were originally drafted by legal counsel, any non-standard changes or any client-proposed modifications need review. Internally, have our legal advisor or contracted attorney review the filled documents if there are significant edits. This is especially true for high-value contracts or those involving unique liability concerns.
4. **Client Signature:** Once finalized, the documents are sent to the client for signature. Typically, the NDA might be signed first (to enable free discussion), then the MSA (with SLA and DPA as needed), and then each SOW. We keep signed copies in our secure document repository (and a hard copy in the project binder possibly).
5. **Ongoing Use:** For subsequent projects with the same client, we usually do not need a new MSA/NDA (unless the original term has lapsed or needs update); we just create a new SOW. Keep track of any expiration dates (some NDAs might have a term for how long info must be kept secret – our template usually says indefinitely for trade secrets, 5 years for other confidences, etc., but ensure no lapse).

All templates have been updated as of 2025 to reflect current regulations and company policies. They contain robust confidentiality clauses reflecting our privacy-centric ethos. For example, the MSA explicitly notes that we **never share client data with third parties** without consent and that we maintain strict security measures (which ties into our security protocols document). This alignment between what we *promise contractually* and what we *implement technically* helps build trust and legal safety.

Confidential Handling: These templates are marked confidential and are for internal use. They should not be given out in raw form to anyone except for the purpose of drafting a client agreement. Always produce a PDF of the completed contract for sending to the client, and do not send them the editable templates. Maintain version control—if any changes are made to the master templates (for instance, by legal counsel), update the vault here and record the change in the change log appendix of this document.

By using these **Client Contract Templates**, we ensure every client engagement is governed by clear, fair, and protective terms. This not only mitigates risk but also reinforces to the client that we operate professionally and value privacy and security in every aspect, including our paperwork.

On-Prem AI Deployment Scripts

This section provides an **automated deployment guide for on-premises AI tools**, specifically focusing on local Large Language Model (LLM) solutions. We supply a set of scripts (packaged in a ZIP archive) that set up the entire stack: **Ollama**, **llama.cpp**, **Open WebUI**, and **LM Studio**. These tools enable running advanced AI models (like LLaMA, GPT-style models) on the client's own hardware, without relying on cloud services. By using our scripts, one can get a fully functioning AI environment quickly, ensuring that all AI computations and data remain on-site (maximizing privacy).

Purpose & Scope: Many clients require AI capabilities (for example, a private GPT-4-style assistant for their data) but cannot send sensitive data to external APIs. Our on-prem AI deployment solves this by installing open-source or self-hosted AI runtime environments. The provided scripts automate installation of the following components:

- **Ollama** – a lightweight CLI tool for running large language models locally. It manages model downloading and provides a simple interface (including an API) for running prompts against local models.
- **llama.cpp** – a foundational backend (in C/C++) that allows efficient inference of LLaMA-family models on CPU (and GPU if available). This is used under the hood by many apps. We include it so that even if other tools fail, you can run models via command-line.
- **Open WebUI** – an open-source web interface for generative AI models. It provides a user-friendly chat UI in a browser, supporting multiple backends (including llama.cpp and Ollama). This allows clients to chat with their local AI model from any PC on the network via a web browser.
- **LM Studio** – a desktop application (with GUI) for running local LLMs, which some users prefer for its simplicity. It's essentially a polished app where users can load models and interact with them without using the command line.

The goal is to deploy all of the above, so the client has flexibility in how they use the AI: programmatically via Ollama, in a web browser via Open WebUI, or in a desktop app via LM Studio. All components will be configured to run **within the client's network, offline**.

Script Functions: The `deploy_onprem_ai.sh` (for example) script included will perform several tasks automatically:

1. **Environment Preparation:** Update the system (for Linux, `apt update && apt upgrade` for instance), install needed dependencies (like Git, Python3, Node.js for some UI perhaps, C++ build tools, etc.). Ensures GPU drivers are present if an NVIDIA GPU is in use for acceleration (and installs CUDA or appropriate libraries when possible).
2. **Install Ollama:** The script uses the official installation method for Ollama. For example, on Linux it would run the one-line installer provided by Ollama (e.g., `curl -sSfL https://ollama.ai/install.sh | sh` which fetches the latest release). After installing, it verifies by running `ollama version`.

3. **Set Up llama.cpp:** The script clones the `llama.cpp` repository from GitHub, compiles it with optimizations (making use of all CPU cores). This yields the `main` binary (and others like `convert.py`). We also download at least one small test model (like LLaMA 2 7B or an open variant) to ensure everything is working. The model files (GGML format) are stored in a designated directory (e.g., `/opt/llm/models`).
4. **Deploy Open WebUI:** The script fetches the Open WebUI project (possibly via Git or a Docker image). We have a preference for using a Docker container for Open WebUI for consistency. The script can install Docker if not present, then pull the `ghcr.io/<open-webui-image>` container. Alternatively, if not using Docker, it sets up a Python virtual environment, installs required packages, and configures the Open WebUI service. After setup, it will configure Open WebUI to know about the models installed (linking it to llama.cpp and Ollama backends).
5. **Install LM Studio:** LM Studio might come as an AppImage or installer. The script will download the latest release of LM Studio (for example, a .deb package or AppImage on Linux, or an MSI on Windows). On a headless server, LM Studio might not run, so this is more for clients who have a GUI system; however, if the server has a desktop environment, we place the LM Studio app for them. The script can also place a shortcut or instructions for connecting to it remotely (like via Remote Desktop or X forwarding).

After installation, the script performs basic configuration: it might create systemd services to auto-start the AI tools on boot. For instance, setting up **Ollama** to run as a background service listening on a certain port for API requests, or launching **Open WebUI** on boot so that the web interface is accessible at `https://<server-ip>:8000` internally. We also configure firewall rules so that these services are only reachable within the LAN or via VPN (no external exposure).

Usage Instructions: The document provides guidance on running the deployment and using the tools:

- To execute the deployment, an admin would copy the `On-Prem AI Deployment Scripts` archive to the target machine (which should be a Linux server or VM with sufficient resources). Unzip the archive, and run the main script: e.g. `bash deploy_llm_stack.sh`. The script will prompt for confirmation and then proceed. It logs its progress to a file (for troubleshooting). After completion, it will indicate success or any issues.
- **Post-Install Testing:** We include steps to verify each component:
- For **Ollama**: run `ollama run llama2` (assuming a model name “llama2” was downloaded) with a sample prompt to see if it returns an answer.
- For **llama.cpp**: run the `./main` binary on a test prompt file to ensure it works in pure CLI mode.
- For **Open WebUI**: open a browser on a client PC and go to the designated URL (the script will output something like `Open WebUI running at http://10.0.0.10:8000` for example). Ensure the interface loads and you can chat with the model.
- For **LM Studio**: if installed on a user workstation, launch it and confirm it detects the model files or can download models to the correct folder (the document offers a brief walkthrough of LM Studio’s UI to load a local model and run a prompt).

We also emphasize resource requirements: Running large models can be hardware-intensive. The guide notes that for a full 70B parameter model, a high-end GPU and >100GB RAM might be needed, whereas smaller 7B or 13B models will run on a modest server (especially with int4 quantization). The script by default might download a smaller model (like a 7B) to ensure functionality, and we include instructions for the admin to download larger models manually if needed (since they might be copyrighted, we point them

to where and how to obtain them, e.g., having the client log in to download LLaMA weights if they have access).

Privacy Considerations: All these AI tools run **completely offline**. None of the installed components will send data to external servers. We even configure them to not automatically check for updates without approval. This ensures that any proprietary or sensitive prompts/data the client uses with these AI models never leave the premises. We mention that maintaining this isolation is crucial: if the client later connects a tool to the internet (for model downloading or some plugin), they should be aware of the data implications. By default, our deployment is locked down – for example, if Open WebUI has a feature to pull community prompts or extensions, we disable it or advise caution.

Maintenance: The guide recommends periodic updates of the AI tools (since the AI field evolves rapidly). We provide a short script or instructions for updating: e.g., how to pull the latest llama.cpp and recompile, how to update Ollama (`ollama pull` if supported), etc. We also instruct how to add new models. The installed system comes with a directory structure (perhaps `/opt/llm_models`) and the user can drop new model files there and update config for Open WebUI or Ollama to register them.

By using the **On-Prem AI Deployment Scripts**, our clients can harness AI capabilities similar to cloud AI services, but entirely within their own secure environment. This empowers them to ask questions, analyze data, or automate tasks with AI while **ensuring data confidentiality** (no queries or responses are exposed to a third-party). The automation saves countless hours in setup and configuration, providing a ready-to-go AI environment with minimal technical effort.

Proxmox GPU Passthrough Guide

This guide offers a step-by-step walkthrough for enabling **GPU passthrough on Proxmox Virtual Environment (VE)**, specifically targeting NVIDIA GPUs (though many steps apply to other GPUs as well). By following this, one can dedicate a physical GPU installed in the Proxmox host to a virtual machine (VM), allowing that VM to use the GPU for graphics or computation as if it were directly attached. This is critical for workloads like AI model inference (as per our AI deployment) or other GPU-accelerated tasks, and even for running a full desktop VM with proper graphics.

1. Verify and Prepare the Hardware:

Ensure the Proxmox host machine supports virtualization extensions (Intel VT-x/VT-d or AMD-V/AMD-Vi for IOMMU). In the system BIOS/UEFI, **enable IOMMU** (often called Intel VT-d or AMD IOMMU) and also enable virtualization for CPU if not already. Enable “Above 4G Decoding” and “Resizable BAR” if those options exist, as they can help especially with GPUs that have large BAR memory (mostly for AMD GPUs or very large VRAM NVIDIA). Physically install the GPU in the server and boot Proxmox.

On the Proxmox host, check that the GPU is detected by the OS: for example, run `lspci | grep -E "VGA|NVIDIA"` and note the device ID (like `01:00.0` for the GPU and maybe `01:00.1` for the GPU’s audio function). Also, ensure no other processes (like a host GUI or a mining service) are using the GPU – our standard Proxmox installs are command-line only, so the GPU should be idle.

2. Proxmox Host Configuration:

We need to configure the Proxmox host to pass the PCI device through to VMs:

- **Edit GRUB for IOMMU:** SSH into the Proxmox host or use the shell. Edit `/etc/default/grub`. Find the `GRUB_CMDLINE_LINUX_DEFAULT` line and append the following parameters (depending on CPU vendor):
- For Intel: `intel_iommu=on iommu=pt`
- For AMD: `amd_iommu=on iommu=pt`
  This ensures IOMMU is active and in passthrough mode. Save the file and then run `update-grub` to apply changes.
- **Load VFIO Modules:** Edit (or create) `/etc/modules` and add these lines:

  ```
  vfio
  vfio_iommu_type1
  vfio_pci
  vfio_virqfd
  ```

  This makes sure the VFIO (Virtual Function I/O) driver is loaded on boot, which is needed to attach devices to VMs.
- **Blacklist Host Drivers:** Since the host doesn't need to use the GPU directly, we prevent the host OS from grabbing it. For NVIDIA, blacklist the nouveau driver (open-source) and possibly the NVIDIA driver if it's installed. To do so, create a file like `/etc/modprobe.d/blacklist-nouveau.conf` with content `blacklist nouveau` (and similarly blacklist `nvidia`, `nvidia_drm`, etc. if any). Additionally, it's recommended to prevent the EFI framebuffer from locking the GPU, by adding `video=efifb:off` to the GRUB cmdline if needed.
- **Bind GPU to VFIO:** Optionally, you can specify that the GPU's IDs should be bound to vfio-pci driver early. One method: create a config at `/etc/modprobe.d/vfio.conf` with:

  ```
  options vfio-pci ids=<GPU_ID>,<Audio_ID>
  ```

  Replace `<GPU_ID>` and `<Audio_ID>` with the vendor:device IDs of your GPU's functions (you can get these from `lspci -n` output, e.g., `10de:1eb8,10de:10f0` format for NVIDIA GPU and its audio). This ensures that on boot, those PCI devices attach to vfio-pci driver instead of any host driver.

After these changes, **reboot the Proxmox host**. When it comes back, verify that the GPU is now using the vfio driver: `lspci -k -s 01:00.0` should show it bound to `vfio-pci` (and not nouveau or nvidia). Also, run `dmesg | grep -e IOMMU -e DMAR` to confirm IOMMU is enabled with no errors.

3. VM Configuration in Proxmox:

Now create or configure a VM to use the GPU:

- If not already done, create a new VM (for example, a Windows 11 VM or an Ubuntu VM depending on need) via the Proxmox web interface. Give it the proper UEFI BIOS (OVMF) if using UEFI, and add the **Qemu Guest Agent** for convenience (especially for Windows).
- **Add the PCI Device:** In the Proxmox web UI, go to the hardware configuration of the VM. Click **Add -> PCI Device**. From the dropdown, select your GPU (it will show up as something like `01:00.0 NVIDIA Corporation ...`). Also, **add the GPU's audio function** (usually `01:00.1`) as a second PCI Device in the VM. When adding, check the boxes for **"PCI-Express"** and **"All Functions"** if available (Proxmox 7/8 allows a checkbox for "All Functions" which ensures multi-function devices like GPUs with audio are handled together).
- **Machine Type and CPU:** It's often recommended to set the VM's BIOS to OVMF (UEFI) and Machine type to `q35` (which supports PCIe passthrough better). For CPU type, use **host** (to expose all CPU features to the VM). And crucially, for an NVIDIA card being passed to a Windows VM, add the following **args** in the VM options (in Proxmox GUI, under VM -> Options -> "args" or via editing `/etc/pve/qemu-server/<vmid>.conf`):

  ```
  args: -cpu host,hidden=1
  ```

  The `hidden=1` part hides the hypervisor signature from the GPU driver; this avoids the infamous "Error 43" that NVIDIA consumer GPUs show in VMs by design. Our guide explains this and ensures it's set.
- **Memory & HugePages:** If the GPU has a lot of VRAM or if doing heavy compute, we advise enabling hugepages for the VM memory. In the VM's Memory settings, you might set "hugepages = 2048" (2MB huge pages) and ensure the host has enough free hugepages. This can improve performance and reduce any memory mapping issues.

4. Install Drivers in VM:

Start up the VM. If it's Windows, on first boot after adding the GPU, it might detect new hardware. You'll need to install the official NVIDIA drivers inside the VM (download from NVIDIA's site). For Linux VMs, install the appropriate NVIDIA proprietary driver package. After installation, reboot the VM. Then verify within the VM that the GPU is working: - On Windows, check Device Manager: the GPU should appear under Display Adapters with no error icon. Run something like GPU-Z or the NVIDIA Control Panel to see that it's functioning. If using it for display, you can attach a virtual monitor or use Parsec/RDP to see GPU output, but if headless for compute, just ensure it's recognized. - On Linux, run `nvidia-smi` in the VM to see if it communicates with the card and shows stats.

5. Testing and Troubleshooting:

We recommend a test workload: e.g., run a quick benchmark or, for AI, try running a model using the GPU (from within the VM). Monitor Proxmox tasks to ensure no errors like IOMMU faults. Common issues and solutions are noted in the guide: - If the VM fails to start and you see an IOMMU error, the device might be in a group that isn't isolated. Sometimes, adding `pch_reset=1` or similar args for certain motherboards is needed, or making sure no other device in the IOMMU group is in use. Our guide suggests using `find /sys/kernel/iommu_groups/ -type l` to see groupings. - Error 43 in Windows device manager means

you likely missed the `hidden=1` flag or NVIDIA drivers think it's virtualized. Double-check VM config. - If the Proxmox host uses UEFI, sometimes the GPU is bound to efifb. We already suggested `video=efifb:off` to grub; if not, do that and reboot. - If you need to free the GPU for host use occasionally, you would unload vfio and load nvidia drivers; but in general, we dedicate the GPU to the VM full-time in our deployments.

6. Advanced:

The guide also briefly covers advanced scenarios: - **Multiple VMs with one GPU:** Not generally possible simultaneously unless using NVIDIA vGPU (not covered here due to licensing) or Intel GVT-g/AMD SR-IOV if supported. Our approach is one GPU per critical VM. If multiple lesser VMs need acceleration, consider time-slicing or an alternate approach (this is not typical in our client setups; usually one powerful VM uses the GPU for AI or a virtual desktop).
- **GPU for Linux acceleration:** If the VM is Linux and headless (for compute), no special steps beyond driver install are needed. If it's for a Linux desktop VM (rare in our case), we ensure Spice/virtio graphics are off and maybe use a dummy HDMI plug on the GPU to trick it if needed for proper resolution.

By following the **Proxmox GPU Passthrough Guide**, our internal team (or the client's IT) can successfully assign a physical GPU to a VM. This unleashes powerful capabilities: for example, a Windows VM can be used for video editing or CAD with GPU, or our Ubuntu server VM can run machine learning tasks at dramatically improved speeds. The process is intricate but our guide breaks it down to manageable steps, ensuring a smooth setup in line with our overall system architecture.

TrueNAS Scale Configuration

This guide provides a complete setup walkthrough for **TrueNAS SCALE**, which is our choice for on-site network-attached storage (NAS) and lightweight container hosting. We detail how to configure ZFS storage pools for optimal performance and reliability, how to set up network shares, and how to deploy applications on TrueNAS SCALE. Following this guide ensures the storage system is rock-solid and tuned for our clients' needs.

1. Installation & Initial Access:

Assuming TrueNAS SCALE is freshly installed on the NAS server (either bare-metal or as a VM with direct disk access), connect to the web UI via a browser (`https://<truenas-ip>/`). The first time setup wizard will prompt for basics: - **Network Config:** Assign a static IP address in the management VLAN (as per the Blueprint). Make sure DNS settings point to the local DNS server (or your router) and that the gateway is set correctly. Enable the web interface over HTTPS for security (TrueNAS can generate a self-signed cert or you can install a proper certificate).
- **Admin Account:** Set a strong root password (our practice: use a generated passphrase, store it in the password manager). Optionally, create a non-root admin user for daily use and disable root login over SSH for security. Ensure SSH is enabled only if needed (and if so, key-based auth is a must).

2. Creating ZFS Pool(s):

TrueNAS SCALE's core is the ZFS filesystem. We need to create a **storage pool** that aggregates all available drives with redundancy: - Navigate to **Storage > Pools > Add**. Choose to create a new pool. Select all the data drives. Our standard configuration for, say, 8 drives might be RAID-Z2 (which can tolerate two drive failures). For 6 drives or fewer, RAID-Z2 is still good; if high performance is needed and capacity trade-off is acceptable, use mirrored pairs. The guide suggests an optimal layout based on drive count (and lists pros/cons).
- **Name the pool** (e.g., `tank1` or `mainpool`). Enable disk encryption if required by client policy (TrueNAS offers dataset-level encryption which we can turn on later; full pool encryption is also possible).
- **Advanced pool options:** We recommend enabling **SSD cache (L2ARC)** if the system has an NVMe or SSD spare to use for read cache and maybe a **ZFS Log device (SLOG)** if we expect a lot of synchronous writes (like for VMs or databases). The guide details: if using an SLOG, it should be a high-endurance NVMe. Also, set **ashift=12** (default for modern drives) to optimize 4K sector alignment. After confirming, TrueNAS will create the pool and all underlying vdevs.

3. Datasets and Organization:

Under the new pool, create **datasets** to organize data and apply specific settings: - We recommend separate datasets for different data types. For example: `tank1/media`, `tank1/backups`, `tank1/vm-disks`, `tank1/config`, etc. Each dataset can have its own properties. For instance, enable compression (lz4, on by default) on all datasets – it's lightweight and saves space. For a dataset that will store large media (videos, etc.), you might turn off deduplication (to save RAM and because media won't dedupe well) and set recordsize to 1M for sequential reads. For a dataset for VMs or databases, you might use a smaller recordsize (16K or 32K) for better I/O matching. The guide provides a table of recommended dataset settings: e.g., - Backups dataset: compression on, recordsize 128K, dedup off. - VM images dataset: compression on, recordsize 16K, sync=standard (or sync=always if using SLOG for safety), dedup off (unless lots of identical VMs). - General documents dataset: compression on, recordsize 128K, maybe enable dedup if the client stores a lot of similar files (use with caution due to high RAM needs). - **Snapshots:** TrueNAS allows easy snapshots. We outline a snapshot policy: e.g., take nightly snapshots of important datasets (and keep, say, last 7 daily, 4 weekly, 12 monthly). The guide shows how to configure periodic snapshot tasks in **Data Protection > Periodic Snapshot Tasks**. These provide on-site versioning (to recover from accidental deletions or ransomware). We explain naming schemes and how to prune old snapshots.

4. Network Shares Setup:

Most clients will need to access the NAS data from other computers. Depending on the environment, set up SMB (for Windows/Mac) or NFS (for Linux or ESXi, etc) shares: - Go to **Sharing**. For a Windows-heavy environment or mixed environment, create an **SMB share** for each dataset that should be shared. The wizard will prompt to enable the SMB service (do so). Provide a name for the share (e.g., "MediaDrive"). Set permissions accordingly: often, we create a group (like "family" or "staff") and assign read/write to that group on the dataset. TrueNAS can set the dataset permissions in the process (choose Windows ACL if using SMB). We emphasize to disable SMB1 (TrueNAS does by default now) and to enable the **recycle bin** option on shares if users want deleted files to be recoverable easily. - For more technical use (like hypervisor storage), set up **NFS shares**. E.g., an NFS share of `tank1/vm-disks` to a Proxmox or VMware host. Add an NFS share, restrict it by network or host (like only allow the Proxmox host's IP to mount it), and map all users to root or nobody as needed depending on use case. - If applicable, mention **AFP/Time Machine** for Mac backups (TrueNAS SCALE doesn't have AFP, but Time Machine can use SMB with the tmutil capability; we guide how to mark a share as a time machine backup target). - We note that all share access will be over

the secure LAN or via VPN – no port forwarding of NAS services to the internet, keeping data access internal only.

5. User Accounts and Access Control:

Under **Accounts**, create user accounts or import directory services: - In a small/home deployment, we manually create users for each person who will access (matching their Windows/Mac logins possibly) and assign them to groups (like a group per share or a general "nasusers" group). Set strong passwords or encourage use of their system credentials (with perhaps Active Directory integration). - In an enterprise setting, TrueNAS can join an Active Directory domain. The guide covers how to do that (under **Credentials > Directory Services**). If the client has AD, join it so that domain users can be granted access to shares without separate NAS credentials. We cover the required info: AD domain name, credentials, setting up an ID mapping, etc. After joining, one can assign domain users/groups permissions on datasets. - Ensure services (SMB especially) are configured to use the directory service for authentication if needed. We mention enabling SMB "fruit" settings if Mac devices for better compatibility, and enabling NFS v4 with Kerberos if using AD for NFS auth (advanced topics given as optional steps). - TrueNAS SCALE can also enable **two-factor auth** for the web UI logins – we suggest enabling that for the root/admin account for extra safety.

6. Apps and Services (TrueNAS SCALE Apps):

One of the powerful features of SCALE is the ability to run containerized apps (through Kubernetes under the hood). We leverage this for additional services: - **Apps Catalog:** Go to **Apps** section. TrueNAS ships with an official catalog and one can add the community TrueCharts catalog. The guide instructs how to add TrueCharts (providing the URL, etc.), which offers many ready-to-deploy apps. - We recommend certain apps as part of our deployment: - **Nextcloud** (or similar) if the client wants a private cloud storage frontend. - **Plex** or **Jellyfin** for media serving (if media is part of the setup). - **Home Automation controllers** (if applicable) like HomeAssistant (though we might use a separate VM, but TrueNAS could run it too). - **Bitwarden (Vaultwarden)** for the password manager (if not running it in Proxmox, TrueNAS app could host it since it's lightweight). - **Syncthing** for syncing files between devices, etc. - Example: Deploy **Vaultwarden (Bitwarden)** from TrueCharts. The guide shows the steps: select the app, set a dataset for it to store data (probably on an SSD pool or a dataset with small recordsize for database), set admin user, etc. Once deployed, the service is available on the network (we'd integrate it with our DNS so e.g. vault.local resolves to it). - Another example: If a client wants to use the NAS for virtualization, TrueNAS SCALE itself can host VMs or Kubernetes pods. However, since we already have Proxmox for heavy VMs, we tend to reserve TrueNAS's VM feature for lightweight utility VMs or to not use it at all. We mention it for completeness but our standard is to let Proxmox handle VMs and TrueNAS handle storage and lightweight containers. - We stress on setting resource limits for apps where appropriate to avoid them consuming too much memory/disk.

7. Performance Optimization:

ZFS tuning and network tuning: - Enable **auto-snapshot pruning** and schedule scrubs monthly (TrueNAS usually auto-schedules scrubs every ~35 days; adjust to ~1 month). Scrubs help detect and heal bit rot with parity, but they are heavy – schedule them for a low usage time. - Check **SMART tests** schedule for disks (we set short tests weekly, long tests monthly). - If the NAS has 10GbE or faster, ensure MTU (jumbo frames) is consistent if using them. Often for simplicity, we stick to 1500 MTU unless a dedicated storage network

allows 9000. The guide notes how to set interface MTU in Network settings if needed. - If using any special vdevs: tune the **ARC size** (ZFS read cache in RAM) via TrueNAS advanced settings if the server also needs to run apps – maybe cap ARC to leave RAM for apps. By default TrueNAS will use a lot of RAM for ARC which is good for pure storage performance. - **Alerts:** TrueNAS has an alert system (disk failures, pool capacity >80%, etc.). Ensure email alert settings are configured (via root email or Alert Services – perhaps sending to our support email or the client's IT). We absolutely want to catch a degraded pool early to swap a disk. - Advise on **capacity management:** Through experience, ZFS pools shouldn't be over 80-90% full for best performance. We mention to monitor usage and plan upgrades if needed. Also, for expansion, we note that ZFS requires adding vdevs (not just single disks easily) so plan for future growth (e.g., leave some drive bays free).

8. Backup and Replication:

We include a section on backup strategies: - If the client has a second TrueNAS or an off-site NAS, we can configure **replication tasks**. TrueNAS can send ZFS snapshots to another TrueNAS over SSH. The guide would show how to set up a periodic replication of critical datasets to an off-site box or to cloud storage (TrueNAS can also sync to cloud, but since we focus on privacy, maybe replication to a physically secure device is preferred). - Additionally, remind about the rotated USB hard drives (if used): If the blueprint included rotating external backups, then outline the procedure: TrueNAS can periodically back up certain datasets to an external USB drive (via Rsync or ZFS send). When a drive is plugged in, an auto-rsync could kick off. This can be configured using autofs and a script or manual process – details beyond scope but concept mentioned.

By meticulously following the **TrueNAS Scale Configuration** guide, our team ensures that the client's NAS is not just a storage box, but a highly reliable data hub: data is organized and protected (ZFS snapshots and parity), accessible but secure (proper shares and permissions), and extensible (apps providing additional functionality on the same hardware). Given that data is the core of many operations, this guide emphasizes both **performance optimization (ZFS tuning)** and **data safety (redundancy, backups)**, aligning with the overall mission of privacy and resilience in our solutions.

WireGuard VPN Automation

This section describes our automated setup for **WireGuard VPN** and the accompanying scripts for client provisioning. WireGuard is a modern, lightweight VPN protocol known for its speed and robust encryption. We use it to provide secure remote access into the client's network (for themselves or for our maintenance). The automation scripts streamline the process of adding new VPN users/clients, generating their configs, and even producing QR codes for easy mobile device setup.

WireGuard Overview: WireGuard operates on the principle of public/private key cryptography. Each peer (server or client) has a key pair. The server (at the client's site, e.g. running on the firewall or a dedicated Raspberry Pi, etc.) listens on a UDP port (usually 51820) and has a list of allowed clients (identified by their public keys). It's ideal for our needs since it's fast (built into the Linux kernel) and has minimal attack surface. We configure WireGuard such that a connected client can access the secure internal network as if they were on-site, which is invaluable for remote monitoring and support, and for clients to connect to their "digital estate" while traveling.

Server Configuration: In our deployments, WireGuard typically runs on the main firewall/router or on a small Linux VM in the DMZ network. The documentation includes the base configuration we use: - The server has a static local IP (often we use a dedicated subnet like 10.100.100.0/24 for VPN). The WireGuard interface on server might be `wg0` with IP 10.100.100.1/24. The server's config lists each client's public key and allowed IP. We set AllowedIPs for each client as their designated VPN IP (e.g., client gets 10.100.100.2/32, etc.) and perhaps their real IP if we want to restrict (though typically not needed). - The server's firewall is configured to allow UDP 51820 from the internet to the WireGuard server. Also, appropriate routing is in place so that traffic from VPN clients (10.100.100.X) can reach the LAN subnets and return. Usually we enable IP forwarding and add static routes if needed or let the server be the gateway for that subnet. Our guide ensures that in the firewall rules, the VPN is treated as just another secure interface: we often restrict VPN clients to only necessary subnets for security (e.g., maybe an installer gets access only to the NAS, while the owner's VPN gets access to everything). - We mention that WireGuard doesn't have a built-in "authenticated" user concept (it's just keys), so losing a key is like losing a password – our procedure to revoke a key is to remove it from config and restart WG. Thus, protecting the keys is important.

Automation Scripts: The key script is something like `wg-auto-provision.sh` which automates adding a new client. What it does: 1. **Generate Keys:** It uses WireGuard's `wg genkey` and `wg pubkey` to create a new private/public key pair for the client. These keys are typically generated on the server (so we can immediately use them in configs). The private key is kept secret and will be put into the client's config; the public key will be added to the server's allowed list. 2. **Update Server Config:** The script will insert a new peer entry into the WireGuard server configuration file (usually `/etc/wireguard/wg0.conf`). That entry includes the new client's public key, an assigned IP (the script chooses the next available IP in the VPN subnet), and AllowedIPs (often just the client's own IP or possibly 0.0.0.0/0 if we route all client traffic through VPN, though in our setups we usually only route internal traffic through VPN, not internet). For example, it might append:

```
[Peer]
# ClientName
PublicKey = <client_public_key>
AllowedIPs = 10.100.100.3/32
```

where 10.100.100.3 is the new client's VPN IP. 3. **Generate Client Config:** The script creates a WireGuard config file for the client (e.g., `ClientName.conf`). This file contains:

```
[Interface]
PrivateKey = <client_private_key>
Address = 10.100.100.3/24
DNS = 10.0.0.1 (if we want to push a DNS, perhaps the local DNS or none)

[Peer]
PublicKey = <server_public_key>
Endpoint = <VPN Internet IP or DDNS>:51820
AllowedIPs = 10.0.0.0/16, 10.100.100.0/24
PersistentKeepalive = 25
```

The AllowedIPs here for the peer section usually include the internal networks that the client should reach via VPN (like the LAN subnet 10.0.0.0/16 in this example, plus the VPN subnet itself). We often do NOT include 0.0.0.0/0 unless we want to route all internet traffic through the VPN (which we typically don't for client convenience; we only provide internal access). `PersistentKeepalive` is set to 25 seconds for roaming clients to maintain NAT holes. 4. **QR Code Generation:** For user convenience, especially when the client will use the VPN on a mobile device (iOS/Android WireGuard app), the script generates a QR code for the new client config. It uses a tool like `qrencode` to convert the client config (which is just text) into a QR image (PNG). This image can be scanned by the WireGuard mobile app to instantly import the config without typing. The script might output the path of the PNG (and perhaps even email it or move it to a known location). We usually mark the QR with the client name and creation date. 5. **Restart/Apply Config:** After adding the peer, the script triggers WireGuard to apply changes. Either by restarting the `wg-quick` interface or by using `wg set wg0 peer <pubkey> allowed-ips <ip>` dynamically. Usually, we opt to just restart the interface quickly (since it's near-instant) or if high availability is needed, use `wg addconf` to add peer without dropping others. The script then confirms that the peer is added (maybe by running `wg show` to list peers).

Usage Example: Suppose we have a new technician that needs access, we'd run:

```
sudo ./wg-auto-provision.sh -n alice_laptop
```

And the script would spit out something like: - "Generated keys for alice_laptop. Assigned IP 10.100.100.3." - "Added peer to wg0.conf and restarted WireGuard." - "Client config saved to /etc/wireguard/clients/alice_laptop.conf" - "QR code image saved to ./alice_laptop_qr.png – scan this with the WireGuard app."

We include in the doc a sample snippet of what a client config looks like (for clarity), e.g.:

```
# Example Client Config (Alice's Laptop)
[Interface]
PrivateKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX=
Address = 10.100.100.3/24
DNS = 10.0.0.10  # internal DNS server

[Peer]
PublicKey = YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY=
Endpoint = vpn.clientnetwork.com:51820
AllowedIPs = 10.0.0.0/16, 10.100.100.0/24
PersistentKeepalive = 25
```

(We make sure not to include actual keys here, just placeholders for documentation.)

Security & Management: The guide stresses that the `wg-auto-provision.sh` script should be run only on the secure server by an authorized admin (it requires root). The script output (especially the private key and config) is sensitive – after providing it to the client (via secure means, e.g., we often print the QR on paper in person or send via an encrypted message), the local copy of the config can be stored in an

encrypted vault or deleted as per policy. We generally keep a copy of each client config in `/etc/wireguard/clients/` which is permission-restricted, so we can reprint or send it if they lose it.

Revocation: If a device is lost or a user no longer should have access, we remove their peer config from the server (and maybe add their key to a revocation list or simply delete it). WireGuard doesn't have a CRL; removal suffices. The guide notes to then restart the interface so that peer can no longer connect. If their config was set to AllowedIPs wider (like 0.0.0.0/0), we also ensure no stray traffic from that IP gets in (though without the key it won't handshake anyway).

Monitoring: We can monitor who's connected by `wg show` which lists latest handshake times for each peer. We instruct the admin to check this if needed. For auditing, one could look at WireGuard logs or use a script that logs handshake events. By default WireGuard is silent (for privacy and simplicity), but we could enable some logging if required. Our environment being small (handful of peers) usually doesn't need advanced logging beyond that.

Integration with our Infrastructure: The VPN IP range (10.100.100.0/24 in example) is part of our network plan. We ensure routes: either the main router knows to route that subnet to the WireGuard server, or if WireGuard is on the main router, it has direct access. This is documented in the blueprint too, but reiterated here: when the VPN is active, a client can, for example, RDP into their on-site PC at 10.0.0.50 or open the NAS UI at 10.0.0.10 as if on local LAN.

We mention that WireGuard can also be used for site-to-site tunnels if needed (not the main focus here, but in enterprise maybe connecting two offices). The script currently is for road warriors (client devices). For site tunnels, config is manual and we treat each site as a peer.

Using the QR Codes: The generated QR code makes mobile onboarding a snap. On iPhone, you open WireGuard app, add tunnel -> scan QR, done. We caution that the QR contains the private key, so it's like a password – don't let others scan it. We provide the PNG to the client securely (if remote, maybe via our secure portal or instruct them to use signal, etc). Sometimes we print it and physically hand it over, then destroy the print after use.

By leveraging these **WireGuard VPN Automation** scripts, we eliminate error-prone manual steps. This ensures every new VPN user is set up with correct config and minimal effort. It also standardizes naming and IP assignment. The result: both our team and clients can securely connect into the network within minutes, enabling remote support, monitoring, and client remote work, all **without compromising on security**. WireGuard's encryption (Curve25519) and minimal codebase give us confidence that our remote access is as safe as the on-premise network, aligning perfectly with our zero-trust and privacy-first philosophy.

Each document above is crafted to be aesthetically polished and thorough. Diagrams, images, and clear formatting are used to make the content as engaging as it is informative. All guidance is tailored to our specific solutions, ensuring consistency across the board. The **Confidential Documentation Vault** serves as the single source of truth for deploying and managing our ultra-secure, private infrastructure offerings – and must be handled with the utmost discretion.
