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
```

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

`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.
