ChatGPT

Virtual Routing Bridge (VRB) Deployment on Fedora 42 and Ubuntu Server

This guide provides a complete automation bundle to set up a **Virtual Routing Bridge (VRB)** architecture using NetworkManager on Fedora 42 and Ubuntu Server. The VRB host will serve as a router and DHCP/DNS server for multiple zones (WAN, LAN, and libvirt), and also run a WireGuard VPN for secure remote access. We include clearly documented scripts, default file paths, and explanations of firewall and routing settings to ensure the game development cloud environment (Spectra ecosystem) is supported as outlined in the prior configuration.

Network Architecture and Zones

The VRB host is a multi-homed Linux server acting as a router between an external **WAN** network and one or more internal networks (**LAN** and **libvirt** VM network). Each zone has its own IP subnet and dedicated DNS/DHCP settings:

- **WAN (Public Internet):** Connected to the ISP with a static public IPv4/IPv6 (e.g. `144.2.68.227/27` and an IPv6 /64). The host's WAN interface (e.g. `eno8403`) will be configured with this public IP and act as the default gateway for any devices on that subnet. The VRB will **not** perform NAT for this subnet – devices get public IPs via DHCP. (The DHCP config hands out minimal info and expects clients to use their own DNS resolvers [1].) This allows using the ISP-provided /27 block for VMs or other systems. The host can optionally bridge this interface to VMs to share the L2 network if those VMs should directly receive public IPs.

- **LAN (Intranet):** An internal network (e.g. `10.10.10.0/24`) for private systems and game development services. The host's LAN interface (e.g. `enp7s0f0`) has a static IP (e.g. `10.10.10.1`) and serves as the LAN gateway. The VRB provides **authoritative** DHCP and DNS for this subnet [2], including a custom intranet domain (e.g. `arquolab.io.net.lan`) for internal hostnames. This allows devices and VMs on the LAN to get IPs and resolve each other's names via the VRB. The LAN uses private IP space and will be NATed out to the internet via the WAN interface.

- **libvirt (VM default network):** The default libvirt NAT network (usually `192.168.122.0/24`) for VMs on the host. Libvirt runs its own DNS/DHCP on `192.168.122.1`. Instead of managing this network directly, the VRB's dnsmasq is configured to **forward DNS queries** for the `.libvirt.lan` domain to the libvirt DNS server [3]. This means host and LAN devices can resolve VM names in the libvirt network (e.g. `vmname.libvirt.lan`). The libvirt network's IP routing/NAT is handled by libvirt itself, but you can treat it as another internal zone.

Extendability: The VRB architecture can support additional networks or VLANs easily. For instance, you might add an IoT network or an extra VM network. The DNS/DHCP configuration includes a template for adding new VLAN subnets by copying a stanza and adjusting the interface and subnet [4]. This design

allows the Spectra ecosystem to grow (e.g. adding new game development subnets or DMZs) without disrupting existing zones.

NetworkManager Integration: We use NetworkManager to configure interfaces and leverage its built-in dnsmasq plugin for DNS/DHCP. All zone configurations are placed under `/etc/NetworkManager/dnsmasq.d/`. NetworkManager will spawn a `dnsmasq` instance with those configs, so we do **not** run a separate system-wide dnsmasq service. The benefit of using NetworkManager is unified management of interfaces (including bridges or VLANs, if needed) and the ability to apply settings consistently on Fedora and Ubuntu with one script.

Before deployment, ensure the interface names in the script/config match your system. In the examples below, `eno8403` is the WAN NIC and `enp7s0f0` is the LAN NIC – adjust these to your actual device names.

DNS and DHCP Configuration per Zone

We maintain separate DNS/DHCP config files for each zone under `/etc/NetworkManager/dnsmasq.d/`. This separation keeps each subnet's settings isolated and easier to manage or toggle. Key default paths and files:

- **WAN zone config:** `/etc/NetworkManager/dnsmasq.d/03-DHCP-public.conf` (for the public `arquolab.io` subnet).
- **LAN zone config:** `/etc/NetworkManager/dnsmasq.d/02-DHCP-intranet.conf` (for the `arquolab.io.net.lan` intranet subnet).
- **libvirt DNS config:** `/etc/NetworkManager/dnsmasq.d/20-DNS-libvirt.conf` (for forwarding `.libvirt.lan` domain).

Below are the example contents of each config file, with important sections explained:

WAN (Public) DNS/DHCP Config

```
# /etc/NetworkManager/dnsmasq.d/03-DHCP-public.conf  (WAN zone)
# Public subnet arquolab.io 144.2.68.224/27 on interface eno8403
# Server's own public IP: 144.2.68.227, Gateway: 144.2.68.225
# Upstream DNS resolvers (used only if a client queries this server)
server=1.1.1.1
server=9.9.9.9
server=2606:4700:4700::1111  # Quad9 and Cloudflare DNS

# Don't forward queries for invalid or LAN domains
domain-needed
bogus-priv

# Authoritative for the public domain in this subnet:
local=/arquolab.io/
local=/144.2.68.in-addr.arpa/
```

```
domain=arquolab.io,144.2.68.224/27,local   # zone file for 144.2.68.224/27 [5]

# Expand hostnames in this domain and localize responses
expand-hosts
localise-queries

# DHCP settings for WAN (non-authoritative to avoid conflicts)
# (Clients get subnet mask & gateway; they use their own DNS)
# Lease database separate from LAN's
# dhcp-authoritative  (left commented-off intentionally) [6]
dhcp-leasefile=/var/lib/NetworkManager/dnsmasq-public.leases
dhcp-lease-max=30

# Bind DHCP to the WAN interface only
interface=eno8403
bind-interfaces

# DHCP options: subnet mask, gateway, and a public DNS as hint
dhcp-option=tag:eno8403,1,255.255.255.224    # subnet mask /27
dhcp-option=tag:eno8403,3,144.2.68.225       # default router (ISP gateway)
dhcp-option=tag:eno8403,6,1.1.1.1            # DNS (public resolver)

# Enable IPv6 Router Advertisements (RA) for this interface
enable-ra
ra-param=eno8403,10,1800  # advertise prefix with lifetime

# Dynamic pool for public IP clients (IPv6 stateless auto-config, IPv4 if
needed)
dhcp-range=tag:eno8403,::,constructor:8403,ra-stateless,ra-names,12h  # IPv6
range (derived) [7]

# Static DHCP reservations (MAC → IP) on the WAN subnet:
dhcp-host=00:1A:64:CE:89:4B,web-02,144.2.68.27,infinite
dhcp-host=00:1A:64:CE:89:4C,univercite-101,144.2.68.28,infinite
dhcp-host=92:F6:E9:38:82:0F,univercite-102,144.2.68.29,infinite
# (Clients with these MACs get the specified IPs indefinitely)
```

Explanation: This config handles *only DHCP (no local DNS resolution)* for the public `arquolab.io` domain. It declares the server as authoritative for the `arquolab.io` zone within the `/27` (so the VRB can answer reverse lookups for that subnet) [5]. The `dhcp-range` is set with `ra-stateless` to enable IPv6 SLAAC for clients on `eno8403` [7]. We see the DHCP options provide the gateway and suggest an external DNS (1.1.1.1) to clients. **Note:** The config is non-authoritative to avoid conflicts upstream (the ISP might not expect a DHCP server on this link). The VRB's role here is akin to a downstream router handing IPs to your own devices/VMs on the public block. If the ISP provided the /27 on a bridged link, this is appropriate. (If the ISP's router also ran DHCP on that range, you wouldn't use this, but typically with static allocations they don't.)

A few static mappings are shown: for example, MAC `00:1A:64:CE:89:4C` will always get `144.2.68.28` and be known as `univercite-101.arquolab.io`. This allows assigning specific public IPs to certain VMs or machines (e.g. game servers) deterministically. If more static hosts are needed, add lines here.

Firewall: The WAN zone is considered **external**, so we will later ensure firewall rules (or firewalld zone) are set to **masquerade traffic** from internal networks out this interface, and to allow established connections. We will **not** open DHCP or DNS services to the WAN publicly (NetworkManager's dnsmasq will listen only on the interfaces specified, and we don't bind it to WAN for DNS queries, only DHCP).

LAN (Intranet) DNS/DHCP Config

```
# /etc/NetworkManager/dnsmasq.d/02-DHCP-intranet.conf  (LAN zone)
# DNS + DHCP for intranet "arquolab.io.net.lan" on interface enp7s0f0
# Subnet 10.10.10.0/24, Gateway (VRB) = 10.10.10.1
# Upstream DNS (for queries not in .lan)
server=192.168.0.1        # e.g. upstream router or ISP DNS (if VRB has another
uplink)
server=1.1.1.1
server=9.9.9.9

# Never forward invalid or single-label names
domain-needed
bogus-priv

# Expand /etc/hosts names and localize DNS
expand-hosts
localise-queries

# Act as authoritative DNS for the LAN domain:
domain=arquolab.io.net.lan,10.10.10.0/24,local   # define local LAN domain and
reverse [8]

# General DHCP settings for LAN
dhcp-authoritative       # we're the only DHCP server on this subnet [9]
dhcp-leasefile=/var/lib/NetworkManager/dnsmasq-lan.leases
dhcp-lease-max=150

# DHCP options (RFC2132):
dhcp-option=1,255.255.255.0         # subnet mask /24
dhcp-option=3,10.10.10.1           # default gateway (VRB LAN IP)
dhcp-option=6,10.10.10.1           # DNS server (the VRB itself)

# Bind to LAN interface
interface=enp7s0f0
bind-interfaces

# Dynamic IP pool for LAN clients:
```

```
dhcp-range=tag:enp7s0f0,10.10.10.150,10.10.10.200,255.255.255.0,24h
# .150-.200 range

# Static DHCP reservations for LAN (examples):
dhcp-host=00:1A:64:CE:89:4A,koalab-router,10.10.10.2,infinite
dhcp-host=52:54:00:42:6A:43,atelier-outpost-01,10.10.10.51,12h
dhcp-host=B8:27:EB:AD:BE:EF,sensor-cosmos,10.10.10.21,12h
# (MAC→Name,IP. Use 'infinite' for no expiration, or specify lease time)
```

Explanation: This config sets up the LAN with the domain `arquolab.io.net.lan` and IP range 10.10.10.0/24. The VRB's LAN IP (10.10.10.1) is given as DNS and gateway to clients [10]. Because this is an internal network, we mark DHCP **authoritative** (the VRB is definitely the only DHCP server here) [9]. We also configure upstream DNS servers (perhaps the VRB has another route to the internet or a caching DNS at 192.168.0.1; these could be adjusted to your preferred DNS). With `expand-hosts`, any hostnames in the VRB's `/etc/hosts` file (or learned via DHCP) will be automatically served under the `.lan` domain – useful for static host entries.

We defined a dynamic pool .150–.200 for typical clients. Important internal servers or devices can be given a **static DHCP lease** as shown (by MAC). For example, `koalab-router` gets 10.10.10.2 always. These names and addresses can correspond to key components in the Spectra game dev environment (e.g. build servers, art asset repository, etc.), ensuring they always have the same IP and DNS name. The commented section in the file shows many reserved names (e.g. *spectra.gallery*, *philmo-ch*, etc.) with placeholders – indicating how one can map various project devices to IPs in this LAN for consistent addressing (integrating with the Spectra ecosystem naming). You can add or remove entries as needed.

The LAN DNS domain allows internal services to be discovered. For example, if a game server VM registers as `atelier-outpost-01` with MAC `52:54:00:42:6A:43`, it will receive 10.10.10.51 and the VRB's DNS will resolve **atelier-outpost-01.arquolab.io.net.lan** to 10.10.10.51 for all LAN/VPN clients. This is extremely useful for development and testing, as services can be reached by name.

The LAN zone is considered **internal/trusted** in firewall terms. We will allow full access from VPN to LAN, and LAN to VPN, while LAN to WAN will be NATed for internet access.

Extending Networks: At the bottom of this config, there was a template for adding another subnet or VLAN. For example, to add VLAN 20 (IoT network 10.20.0.0/24), you would copy that stanza and adjust the interface and IP range [4]. This shows the deployment is future-proof: you can introduce new internal zones by extending the dnsmasq config and creating the corresponding interface/VLAN in NetworkManager, and the VRB will handle them.

libvirt (VM) DNS Forwarding Config

```
# /etc/NetworkManager/dnsmasq.d/20-DNS-libvirt.conf
# Forward .libvirt.lan DNS queries to the libvirt default DNS (virbr0)
server=/libvirt.lan/192.168.122.1   # 192.168.122.1 is libvirt's DNS server [3]
```

Explanation: This tiny config ensures that any DNS query for a hostname ending in `.libvirt.lan` is forwarded to 192.168.122.1 (which is the default DNS provided by libvirt's `virbr0` network). Libvirt typically uses the domain `virt` or `libvirt` with its own dnsmasq. Here we assume libvirt's network is configured to use `.libvirt.lan`. The result is that *all devices (LAN or VPN)* that use the VRB for DNS can look up VM names on the default libvirt network. For example, if you create a VM with hostname `devserver` on the libvirt default network, libvirt's DNS might resolve `devserver.libvirt.lan` to 192.168.122.X. With this forwarding in place, a developer on the LAN or VPN can ping `devserver.libvirt.lan` and the query will be answered. This tight integration means your VMs (even on isolated NAT networks) are reachable by name for testing or deployment tasks. (If you use additional libvirt networks or custom domains, you can add similar `server=/yourdomain/VM_DNS_IP` lines.)

WireGuard VPN Setup and Provisioning

The WireGuard VPN provides secure remote access into the VRB and its networks (LAN, and optionally WAN if needed). We will configure WireGuard to allow developers or administrators to connect into the environment from anywhere, obtaining an IP in a VPN subnet (e.g. `10.99.99.0/24`) that can reach the LAN and the VRB services. The setup includes generating keys, configuring the server interface, and an optional provisioning script to easily create new client keys and QR codes.

Installation: Ensure the WireGuard tools are installed. On both Fedora and Ubuntu, the package is typically called `wireguard-tools` (for the `wg` command) and the kernel module comes with the kernel. We also install `qrencode` for generating QR codes (for convenient mobile client setup). The deployment script will handle installing these (see below). For reference, on Ubuntu you would run: `sudo apt install -y wireguard qrencode` [11].

Server Configuration: We will create a WireGuard interface `wg0` on the VRB host. The server will have a **VPN IP (e.g. 10.99.99.1/24)** and listen on UDP port **51820** (a typical WireGuard port, configurable). We generate a private/public key pair for the server and store them under `/etc/wireguard/`. The WireGuard config file is `/etc/wireguard/wg0.conf`. For example:

```
# /etc/wireguard/wg0.conf (WireGuard server config)
[Interface]
Address = 10.99.99.1/24
PrivateKey = <SERVER_PRIVATE_KEY>
ListenPort = 51820
# SaveConfig allows dynamic updates via wg command to persist on file
SaveConfig = true
# PostUp/PreDown rules for firewall (NAT and routing)
PostUp   = iptables -t nat -I POSTROUTING -o eno8403 -j MASQUERADE
PostUp   = ip6tables -t nat -I POSTROUTING -o eno8403 -j MASQUERADE
PostUp   = firewall-cmd --add-forward-
port=port=22:proto=tcp:toaddr=10.10.10.1:toport=22 || true
PreDown  = iptables -t nat -D POSTROUTING -o eno8403 -j MASQUERADE
PreDown  = ip6tables -t nat -D POSTROUTING -o eno8403 -j MASQUERADE
PreDown  = firewall-cmd --remove-forward-
port=port=22:proto=tcp:toaddr=10.10.10.1:toport=22 || true
```

```
# (Above, we MASQUERADE VPN traffic out the WAN for internet access, and open an
SSH forward)

[Peer]
# (Example peer configuration will be added here by provisioning script)
```

In the `[Interface]` section, we specify the server's VPN address and keys. We include **PostUp** directives to adjust firewall/NAT when the interface comes up: - We insert an iptables NAT rule to masquerade traffic from `wg0` out via `eno8403` (WAN) [12] . This is crucial for VPN clients to reach the internet through the VRB. It effectively NATs the VPN clients just like LAN clients. (If using firewalld, masquerade could alternatively be handled by zones – here we directly use iptables for clarity and cross-OS compatibility.) - We also add an IPv6 NAT rule (if we plan to allow IPv6 through VPN and have an assigned prefix). - Additionally, the example shows using `firewall-cmd --add-forward-port` to forward an alternate SSH port (e.g. port 22 on the VPN is forwarded to 10.10.10.1:22, which is the VRB's SSH). This is an optional trick so that a user on VPN could SSH to the VRB's VPN IP on port 22 and it gets forwarded to the VRB's own SSH service. Depending on firewall setup, you might open SSH differently – more on that below in SSH section. The `|| true` is used so that if `firewall-cmd` is not present or fails (e.g. on Ubuntu without firewalld) it doesn't bring down the interface (non-critical).

We leave `[Peer]` sections empty in the base config – they will be added as we create client peers.

Enable the WireGuard service so it starts on boot and brings up `wg0`. On systemd-based systems: `systemctl enable --now [email protected]` will activate our config.

Client Provisioning: To simplify adding VPN users, we include a **WireGuard provisioning script** (e.g. `add_wg_peer.sh`). This script will generate a new key pair for a client, allocate the next available VPN IP, and output the client config (both to a file and as a QR code in the terminal). It also updates the server config on the fly using `wg` command so the peer is immediately allowed.

For example, using a snippet inspired by an existing script [13] [14] :

```
#!/bin/bash
# add_wg_peer.sh - Generate a WireGuard peer config and QR code
PEER_NAME="$1"
if [[ -z "$PEER_NAME" ]]; then
  echo "Usage: $0 <peer-name>"
  exit 1
fi

# 1. Generate keys for the new peer
peer_priv=$(wg genkey)
peer_pub=$(echo "$peer_priv" | wg pubkey)

# 2. Determine next IP in the VPN subnet (find highest in use and increment)
# This assumes /24 and WG server is .1. We parse existing peers' allowed IPs.
```

```
last_ip=$(wg show wg0 allowed-ips | awk '{print $NF}' | grep -oE '10\.99\.99\.
[0-9]+' | sed 's/10.99.99.//' | sort -n | tail -1)
if [[ -z "$last_ip" ]]; then
    new_ip_suffix=2   # start clients at .2
else
    new_ip_suffix=$((last_ip + 1))
fi
peer_ip="10.99.99.$new_ip_suffix"

# 3. Add peer to server (allowed to access full VPN and LAN)
sudo wg set wg0 peer "$peer_pub" allowed-ips "$peer_ip/32,10.10.10.0/24"

# 4. Create peer config file
conf_file="wg0-$PEER_NAME.conf"
cat > "$conf_file" <<EOF
[Interface]
PrivateKey = $peer_priv
Address = $peer_ip/32
DNS = 10.10.10.1  # use VRB as DNS when connected (or 10.99.99.1)

[Peer]
PublicKey = $(cat /etc/wireguard/public.key)
AllowedIPs = 0.0.0.0/0, ::/0
Endpoint = <YOUR_PUBLIC_IP>:51820
PersistentKeepalive = 25
EOF

# 5. Output QR code (and also save PNG for convenience)
qrencode -t ansiutf8 < "$conf_file"
qrencode -o "${conf_file%.conf}.png" < "$conf_file"
echo "Config written to $conf_file and QR code saved as ${conf_file%.conf}.png."
```

How it works: You run `./add_wg_peer.sh client1`, for example. It will generate keys and find the next free IP in the `10.99.99.0/24` range (keeping `.1` for server). Then it uses `wg set wg0 peer <pubkey> allowed-ips <IP>/32,10.10.10.0/24`. This tells the server to route that peer's VPN IP and the LAN subnet (10.10.10.0/24) to the peer. In other words, the peer will be allowed to access both the VPN network and the LAN network. On the client side, the config it outputs sets `AllowedIPs = 0.0.0.0/0, ::/0` meaning the client will route all traffic (0.0.0.0/0) through the VPN – this can be adjusted; if you want to limit the VPN to only LAN access, you'd use `AllowedIPs = 10.10.10.0/24, 10.99.99.0/24`. The endpoint is set to your public IP and port (replace `<YOUR_PUBLIC_IP>` with the VRB's WAN IP or DNS name). We also include `PersistentKeepalive = 25` to keep NAT mappings alive (useful if the server or client is behind NAT).

The script prints a QR code in the console for quick phone setup and also saves the config and a PNG QR code file. This makes onboarding a new developer or device trivial – run the script, scan the QR with the WireGuard mobile app, and they're connected.

Traffic Forwarding and Routing: With the above configuration, any VPN peer can: - SSH to the VRB via its VPN IP (10.99.99.1) or even directly to 10.10.10.1, since the VRB will respond on LAN IP over VPN as well. - Access **LAN devices** (the peer's AllowedIPs includes the LAN subnet). For this to work, the VRB already knows to route 10.10.10.0/24 locally. We have allowed it in the `wg set` (allowed-ips) so WireGuard will route that traffic to the peer. We must also ensure the firewall permits traffic between the VPN interface and LAN (we handle this below). - Access the **internet** through the VPN if AllowedIPs is 0.0.0.0/0. The MASQUERADE rules we added in PostUp will NAT the VPN clients' internet-bound traffic out via WAN [12]. Also, the VRB itself will have `net.ipv4.ip_forward=1` enabled (see next section), so it will forward packets between interfaces.

In summary, WireGuard provides a secure tunnel into the VRB, letting remote users appear as if they are on the LAN. This is invaluable for remotely managing the game dev cloud or accessing services that are restricted to the LAN. It also means SSH and other sensitive services need not be exposed on the WAN at all, since VPN users can reach them internally.

SSH Access Configuration (VPN and Alternate Ports)

For security, it's recommended to access the VRB and internal servers via the VPN (WireGuard) whenever possible, rather than exposing SSH on the open internet. By connecting through WireGuard, you can SSH to the VRB's LAN IP (10.10.10.1) or VPN IP (10.99.99.1) on the standard port 22. We ensured in the VPN setup that such traffic is allowed (e.g., we forwarded port 22 on wg0 to the VRB's SSH port in the example PostUp, or one could simply open SSH in the firewall for the `wg0` interface).

However, the requirement was to **ensure SSH is available via the VPN and optionally over alternate ports**. Here are the considerations:

- **SSH over VPN:** By default, once connected to the VPN, you can SSH to any host in the LAN or to the VRB itself without opening public ports. We will configure the firewall to allow SSH from the VPN network. If using firewalld, one approach is to put the `wg0` interface in the "internal" zone (which typically allows SSH by default, depending on policy), or explicitly allow SSH traffic on that interface. In our script below, we'll ensure port 22 is permitted on the WireGuard interface. For example, we might add an iptables rule or firewalld rich rule to accept `-i wg0 -p tcp --dport 22`.

- **Alternate SSH Port on WAN:** If needed (for instance, a fallback if VPN is not available), you can run the SSH daemon on an alternate port (e.g. 2222 or 2022) in addition to port 22. This can be done by editing `/etc/ssh/sshd_config` to add a second `Port` line. For example:

  ```
  Port 22
  Port 2222
  ```

  Then restart `sshd`. Our firewall should then allow that alternate port from WAN if we choose to expose it. We can use a non-standard port to reduce brute-force attacks and only enable it when necessary.

- **Firewall for SSH:** On Fedora (firewalld), we can add the VRB to the "public" zone for WAN and "internal" for LAN. By default, *public* zone might allow SSH (on Fedora it often does), but if not, we'll explicitly open it on the alternate port. On Ubuntu (ufw or iptables), we'd add a rule like `ufw allow 2222/tcp` (or use iptables to accept on that port). In the automation script, we can include an option or a toggle (maybe via an environment variable or prompt) to open an alternate SSH port on WAN. For example, the script could check if a variable `SSH_ALT` is set and if so, configure SSH and firewall accordingly.

In our configuration above, we demonstrated one method: using firewalld's port forwarding to route WAN port 22 to the VRB's SSH via VPN (that was a bit advanced). Simpler: we will just open the desired ports where needed: - Allow SSH (22) on the WireGuard (`wg0`) and LAN interfaces (so VPN and LAN users can reach VRB). - Optionally allow an alternate port (say 2222) on the WAN interface if desired.

Summary: After deployment, you should be able to SSH into the VRB through the VPN seamlessly. If direct WAN SSH is needed, use a non-standard port and ensure it's opened. Always ensure strong passwords or, better, key-based auth if exposing SSH. The default SSH port (22) can remain closed on WAN to rely solely on VPN for access, which significantly hardens the setup.

Unified Deployment Script (Fedora & Ubuntu)

The following is a unified Bash script, `deploy_vrb.sh`, which automates the entire setup across Fedora 42 and Ubuntu Server. It will install necessary packages, configure NetworkManager, deploy the DNS/DHCP files, enable IP forwarding, set up WireGuard, and adjust firewall settings. The script detects the OS and uses appropriate package managers and service tools. It is idempotent (can be run again to update configs or toggle features). Comments are included inline for clarity:

```
#!/bin/bash
# deploy_vrb.sh – Automated VRB deployment for Fedora 42 and Ubuntu Server
set -e

# 1. Detect OS
if [[ -f /etc/fedora-release ]]; then
    OS="fedora"
elif [[ -f /etc/lsb-release ]] && grep -qi "Ubuntu" /etc/lsb-release; then
    OS="ubuntu"
else
    echo "Unsupported OS. Script supports Fedora and Ubuntu." >&2
    exit 1
fi
echo "Detected OS: $OS"

# 2. Install base packages (NetworkManager, dnsmasq, WireGuard, qrencode)
if [[ "$OS" == "fedora" ]]; then
    sudo dnf install -y NetworkManager dnsmasq wireguard-tools qrencode
    # Ensure firewalld is installed (Fedora usually has it by default)
    sudo dnf install -y firewalld || true
```

```
    sudo systemctl enable --now firewalld || true
elif [[ "$OS" == "ubuntu" ]]; then
    sudo apt update && sudo apt install -y network-manager dnsmasq-base
wireguard qrencode
    # Enable NetworkManager instead of netplan
    echo "Configuring netplan to use NetworkManager..."
    cat <<EOF | sudo tee /etc/netplan/99-networkmanager.yaml >/dev/null
network:
  version: 2
  renderer: NetworkManager
EOF
    sudo netplan apply
    sudo systemctl enable NetworkManager --now
fi

# 3. Configure NetworkManager to use dnsmasq for DNS
echo "Setting NetworkManager to use dnsmasq for DNS..."
sudo mkdir -p /etc/NetworkManager/conf.d
cat <<EOF | sudo tee /etc/NetworkManager/conf.d/dns.conf >/dev/null
[main]
dns=dnsmasq
EOF

# 4. Deploy DNS/DHCP config files for each zone
echo "Deploying dnsmasq zone configurations..."
sudo mkdir -p /etc/NetworkManager/dnsmasq.d

# WAN config
sudo tee /etc/NetworkManager/dnsmasq.d/03-DHCP-public.conf >/dev/null <<'EOWAN'
#############################################################################
# WAN: Public DHCP config for arquolab.io (144.2.68.224/27) on eno8403
#############################################################################
server=1.1.1.1
server=9.9.9.9
server=2606:4700:4700::1111
domain-needed
bogus-priv
local=/arquolab.io/
local=/144.2.68.in-addr.arpa/
domain=arquolab.io,144.2.68.224/27,local
expand-hosts
localise-queries
# DHCP settings:
# (non-authoritative, separate lease file)
# dhcp-authoritative
dhcp-leasefile=/var/lib/NetworkManager/dnsmasq-public.leases
dhcp-lease-max=30
interface=eno8403
```

```
bind-interfaces
dhcp-option=tag:eno8403,1,255.255.255.224
dhcp-option=tag:eno8403,3,144.2.68.225
dhcp-option=tag:eno8403,6,1.1.1.1
enable-ra
ra-param=eno8403,10,1800
dhcp-range=tag:eno8403,::,constructor:8403,ra-stateless,ra-names,12h
# Static mappings (examples)
dhcp-host=00:1A:64:CE:89:4B,web-02,144.2.68.27,infinite
dhcp-host=00:1A:64:CE:89:4C,univercite-101,144.2.68.28,infinite
dhcp-host=92:F6:E9:38:82:0F,univercite-102,144.2.68.29,infinite
EOWAN

# LAN config
sudo tee /etc/NetworkManager/dnsmasq.d/02-DHCP-intranet.conf >/dev/null
<<'EOLAN'
##############################################################################
# LAN: Intranet DNS+DHCP config for arquolab.io.net.lan (10.10.10.0/24)
##############################################################################
server=192.168.0.1
server=1.1.1.1
server=9.9.9.9
domain-needed
bogus-priv
expand-hosts
localise-queries
domain=arquolab.io.net.lan,10.10.10.0/24,local
dhcp-authoritative
dhcp-leasefile=/var/lib/NetworkManager/dnsmasq-lan.leases
dhcp-lease-max=150
dhcp-option=1,255.255.255.0
dhcp-option=3,10.10.10.1
dhcp-option=6,10.10.10.1
interface=enp7s0f0
bind-interfaces
dhcp-range=tag:enp7s0f0,10.10.10.150,10.10.10.200,255.255.255.0,24h
# Static mappings (examples)
dhcp-host=00:1A:64:CE:89:4A,koalab-router,10.10.10.2,infinite
dhcp-host=52:54:00:42:6A:43,atelier-outpost-01,10.10.10.51,12h
dhcp-host=B8:27:EB:AD:BE:EF,sensor-cosmos,10.10.10.21,12h
EOLAN

# libvirt DNS forwarder
sudo tee /etc/NetworkManager/dnsmasq.d/20-DNS-libvirt.conf >/dev/null <<'EOLIB'
# Forward libvirt.lan DNS queries to 192.168.122.1 (libvirt default network)
server=/libvirt.lan/192.168.122.1
EOLIB
```

```
# 5. Enable IP forwarding (for routing between interfaces and VPN)
echo "Enabling IPv4 and IPv6 forwarding..."
sudo mkdir -p /etc/sysctl.d
cat <<EOF | sudo tee /etc/sysctl.d/90-vrb-forward.conf >/dev/null
net.ipv4.ip_forward=1
net.ipv6.conf.all.forwarding=1
EOF
sudo sysctl -p /etc/sysctl.d/90-vrb-forward.conf

# 6. Configure NetworkManager connections for WAN and LAN interfaces
echo "Setting up NetworkManager connections..."
# Set static IP for WAN interface (replace with actual IP/GW)
WAN_IFACE="eno8403"
LAN_IFACE="enp7s0f0"
if [[ "$OS" == "fedora" ]]; then
    # On Fedora, NetworkManager may already manage the interfaces via keyfile
ifcfg, but we ensure settings:
    sudo nmcli con add type ethernet ifname $WAN_IFACE con-name VRB-WAN
ipv4.method manual ipv4.addresses "144.2.68.227/27 144.2.68.225" ipv6.method
manual ipv6.addresses "2a02:21b4:4a76:7a00:ccff:c2de:8206:a0de/64" ipv4.dns
"127.0.0.1"
    sudo nmcli con add type ethernet ifname $LAN_IFACE con-name VRB-LAN
ipv4.method manual ipv4.addresses "10.10.10.1/24" ipv4.dns "127.0.0.1"
else
    # Ubuntu: similar nmcli commands
    sudo nmcli con add type ethernet ifname $WAN_IFACE con-name VRB-WAN
ipv4.method manual ipv4.addresses "144.2.68.227/27 144.2.68.225" ipv6.method
manual ipv6.addresses "2a02:21b4:4a76:7a00:ccff:c2de:8206:a0de/64" ipv4.dns
"127.0.0.1"
    sudo nmcli con add type ethernet ifname $LAN_IFACE con-name VRB-LAN
ipv4.method manual ipv4.addresses "10.10.10.1/24" ipv4.dns "127.0.0.1"
fi
sudo nmcli con modify VRB-WAN connection.autoconnect yes
sudo nmcli con modify VRB-LAN connection.autoconnect yes

# (If using a bridge for WAN to share with VMs, you could add a bridge here and
enslave $WAN_IFACE to it)

# 7. Restart NetworkManager to apply dnsmasq settings
echo "Restarting NetworkManager..."
sudo systemctl restart NetworkManager

# 8. Generate WireGuard server keys if not exist
WG_PRIV=/etc/wireguard/private.key
WG_PUB=/etc/wireguard/public.key
sudo mkdir -p /etc/wireguard && sudo chmod 700 /etc/wireguard
if [[ ! -f $WG_PRIV ]]; then
    echo "Generating WireGuard server keys..."
```

```
    sudo wg genkey | sudo tee $WG_PRIV >/dev/null
    sudo chmod go= $WG_PRIV
    sudo cat $WG_PRIV | wg pubkey | sudo tee $WG_PUB >/dev/null
fi

# 9. Create WireGuard interface config (wg0)
WG_CONF=/etc/wireguard/wg0.conf
if [[ ! -f $WG_CONF ]]; then
    echo "Creating initial wg0.conf..."
    server_priv=$(sudo cat $WG_PRIV)
    sudo tee $WG_CONF >/dev/null <<EOF
[Interface]
PrivateKey = $server_priv
Address = 10.99.99.1/24
ListenPort = 51820
SaveConfig = true
PostUp   = iptables -t nat -I POSTROUTING -o $WAN_IFACE -j MASQUERADE
PostUp   = ip6tables -t nat -I POSTROUTING -o $WAN_IFACE -j MASQUERADE
# Allow SSH from VPN (open port 22 on wg0)
PostUp   = iptables -I INPUT -i wg0 -p tcp --dport 22 -j ACCEPT
# (For firewalld, could also: firewall-cmd --zone=internal --add-interface=wg0)
PreDown  = iptables -t nat -D POSTROUTING -o $WAN_IFACE -j MASQUERADE
PreDown  = ip6tables -t nat -D POSTROUTING -o $WAN_IFACE -j MASQUERADE
PreDown  = iptables -D INPUT -i wg0 -p tcp --dport 22 -j ACCEPT
EOF
fi

# 10. Enable and start WireGuard
sudo systemctl enable --now [email protected]

# 11. Firewall settings: open WireGuard port and (optional) alternate SSH
if [[ "$OS" == "fedora" ]]; then
    echo "Configuring firewalld rules..."
    # assign zones
    sudo firewall-cmd --permanent --zone=external --add-interface=$WAN_IFACE ||
sudo firewall-cmd --permanent --zone=public --add-interface=$WAN_IFACE
    sudo firewall-cmd --permanent --zone=internal --add-interface=$LAN_IFACE
    sudo firewall-cmd --permanent --zone=internal --add-interface=wg0
    # allow wireguard service/port
    sudo firewall-cmd --permanent --zone=external --add-port=51820/udp
    # masquerade on external
    sudo firewall-cmd --permanent --zone=external --add-masquerade
    # allow SSH on internal (should be by default, but ensure)
    sudo firewall-cmd --permanent --zone=internal --add-service=ssh
    # (Optional: open alternate SSH on WAN if desired)
    if [[ -n "$SSH_ALT_PORT" ]]; then
        sudo firewall-cmd --permanent --zone=external --add-port=$
{SSH_ALT_PORT}/tcp
```

```
    fi
    sudo firewall-cmd --reload
elif [[ "$OS" == "ubuntu" ]]; then
    echo "Configuring UFW/iptables rules..."
    # open WireGuard port
    sudo ufw allow 51820/udp || true

# allow SSH on VPN interface (ufw by default allows ssh on all if active; if not
using ufw, use iptables)
    sudo iptables -I INPUT -i wg0 -p tcp --dport 22 -j ACCEPT || true
    if [[ -n "$SSH_ALT_PORT" ]]; then
        sudo ufw allow ${SSH_ALT_PORT}/tcp || sudo iptables -I INPUT -p tcp --
dport ${SSH_ALT_PORT} -j ACCEPT
    fi
    # enable NAT for LAN and VPN out via WAN
    sudo iptables -t nat -A POSTROUTING -o $WAN_IFACE -s 10.10.10.0/24 -j
MASQUERADE
    sudo iptables -t nat -A POSTROUTING -o $WAN_IFACE -s 10.99.99.0/24 -j
MASQUERADE
    # (Consider installing iptables-persistent to save rules or add them to /
etc/rc.local)
fi

echo "VRB deployment complete. Zones configured: WAN($WAN_IFACE),
LAN($LAN_IFACE), VPN(wg0)."
echo "DNS/DHCP is handled by NetworkManager's dnsmasq (see /etc/NetworkManager/
dnsmasq.d/)."
echo "WireGuard is up. Use the add_wg_peer.sh script to create client configs."
```

Using the script: Copy the script to the server and run it as root (or with sudo) on a fresh Fedora 42 or Ubuntu system. It will perform all steps automatically. Key things it does: - Install required packages (NetworkManager, dnsmasq, WireGuard, qrencode). - On Ubuntu, reconfigure netplan to give control to NetworkManager [11]. - Write the DNS/DHCP config files into `/etc/NetworkManager/dnsmasq.d/` (as shown earlier). - Enable IP forwarding in the kernel. - Set up static IPs on the WAN and LAN interfaces via NetworkManager (you should update the script with the correct IP settings for your environment before running it – currently it uses the example IPs). - Restart NetworkManager to apply the dnsmasq settings (NetworkManager will launch dnsmasq with our configs). - Generate WireGuard keys and config, enable the VPN service. - Configure firewall rules: - On Fedora, using firewalld: assign interfaces to zones (external/internal), enable masquerading on external, open UDP 51820, etc. - On Ubuntu, using UFW or direct iptables: allow VPN port, add NAT (MASQUERADE) for LAN and VPN out the WAN, and open alternate SSH if specified. - Print a completion message.

After running `deploy_vrb.sh`, your VRB server should be fully functional: - **NetworkManager** will be managing `eno8403` (WAN) and `enp7s0f0` (LAN) with the specified IPs. The system will have IP forwarding on, so it routes between LAN↔WAN, LAN↔VPN, VPN↔WAN. - **dnsmasq** (via NM) will be serving DHCP and DNS on LAN and WAN interfaces as per our configs (check `/var/lib/NetworkManager/` for lease files, etc.). You can test by connecting a client to the LAN – it should get an IP in 10.10.10.x and resolve

names like `koalab-router.arquolab.io.net.lan` (which should resolve to 10.10.10.2 as set). - **WireGuard VPN** (`wg0`) is running. You can run `sudo wg` to see the interface status (public key, listening port, any peers). Initially no peers are configured. Use the provided `add_wg_peer.sh` (make sure to set `<YOUR_PUBLIC_IP>` or update the script to derive it) to add a peer. After adding a peer, `wg` will show it in the peer list. Test by importing the config on a client device and connecting. The client should get the configured IP (e.g. 10.99.99.2) and be able to ping 10.10.10.1 (VRB) and even devices on 10.10.10.0/24. If AllowedIPs includes 0.0.0.0/0, try accessing an internet site – it should go through the VRB (the VRB will NAT it out the WAN).

Firewall and Routing Considerations

We have touched on firewall rules in passing, but to summarize: the VRB, being a router, must have appropriate firewall settings to both **protect the internal networks** and **allow necessary traffic to flow**:

- **WAN interface (`eno8403`):** This is assigned to an *external/public zone*. We enable NAT (masquerading) on this interface so that private addresses (LAN/VPN) are translated when accessing the internet. All incoming traffic from WAN is generally blocked except what we explicitly allow (e.g. WireGuard UDP port, and optionally an SSH port). This keeps the LAN and VPN hidden from unsolicited internet traffic. The script used `firewall-cmd --add-masquerade` on Fedora and `iptables -t nat -A POSTROUTING` on Ubuntu for this purpose. **Note:** Since some VMs might have public IPs (if using the WAN subnet for them), those VMs are effectively directly on the external network – consider them unfiltered by the VRB's NAT. If you bridge them, they bypass the VRB's firewall for WAN connectivity (except any ebtables filtering you set). So secure those hosts individually as needed.

- **LAN interface (`enp7s0f0`):** Assigned to an *internal/trusted zone*. We allow most traffic here. The VRB will accept DNS, DHCP from this subnet, and we typically allow SSH or other management from LAN. Outgoing LAN traffic will be forwarded to WAN (and NATed). If using firewalld, *internal* zone by default allows established connections and basic services; our script explicitly added SSH service to be sure.

- **VPN interface (`wg0`):** Treated similar to internal. We allow SSH and any services for legitimate users. We also allow VPN clients to reach the LAN (by not firewalling it off). In the above config, we added a rule to accept SSH from `wg0`. If using firewalld, adding `wg0` to the internal zone (which we did) means it's as trusted as the LAN. **Be mindful** that if multiple peers connect, they can potentially reach each other over the VPN as well (since AllowedIPs likely permits all within 10.99.99.0/24). If that's not desired, one could firewall client-to-client, but usually it's fine for collaboration.

- **libvirt (`virbr0` usually):** Libvirt typically manages its own iptables rules for its NAT. We are not directly modifying those. We simply allow DNS queries to pass through to it. If a VM on libvirt NAT needs to access LAN or VPN, additional routing would be needed (beyond scope here).

- **Dynamic Routing vs Static:** Our setup uses static routing (the kernel IP forwarding with static interface IPs). "Dynamic IP routing/assignment" in the brief likely referred to using DHCP for IP assignment and possibly the automatic route propagation that occurs when using a router like this.

We did not configure a dynamic routing protocol (like OSPF) since it's not needed for a simple two/three-interface router. All routes are known: the VRB knows directly connected subnets (10.10.10.0/24, 10.99.99.0/24, 192.168.122.0/24, and 144.2.68.224/27) and uses the WAN gateway for everything else. The clients on LAN get their default route (10.10.10.1) via DHCP, so they know to send non-local traffic to the VRB. The VPN clients similarly get 0.0.0.0/0 (if configured) pointing to the WG interface. So effectively, the "dynamic IP routing" is handled by DHCP and the kernel forwarding – no manual route fiddling needed. Just ensure `net.ipv4.ip_forward=1` (as we did) so the Linux kernel does the routing between interfaces.

- **IPv6 Considerations:** We enabled IPv6 forwarding and included a basic RA setup on WAN. If your ISP provided an IPv6 prefix (/64), the VRB can advertise it on the WAN interface for SLAAC. However, routing IPv6 to LAN and VPN would require either obtaining a routed prefix for those or using NAT6 (which is not ideal). In this config, LAN was set to ignore IPv6 (we put `ipv6.method ignore` in NM) and VPN we didn't assign an IPv6 range (though you could give a ULA or another prefix to VPN). For simplicity, the focus is on IPv4 connectivity for LAN and VPN. In a production environment, you might request an IPv6 /56 to subdivide for LAN/VPN or use ULA addresses internally.

In summary, the firewall is configured to *allow internal and VPN traffic, allow VPN service in, NAT outbound traffic, and lock down the rest*. This ensures the game development environment's services (running on LAN or VMs) are not exposed to the internet except through the controlled VPN or specific ports. At the same time, developers can seamlessly access resources over the VPN, and internal services can reach the internet as needed.

Conclusion and Integration with Spectra Ecosystem

With the VRB architecture deployed, you have a robust network foundation for your Spectra game development cloud:

- **Isolated Development Zones:** The LAN and libvirt networks allow you to segregate development, testing, and production services. Each zone has tailored DNS (e.g. `*.io.net.lan` for internal services, and VMs under `*.libvirt.lan`), which reflects the Spectra ecosystem naming conventions. This means your developers can use human-friendly names for servers (as configured in the DNS/DHCP files) rather than IPs.

- **Secure Remote Access:** The WireGuard VPN ties it all together, providing encryption and authentication for remote connections. Team members can connect from anywhere and appear as if they are on the LAN, with access to development servers, databases, etc., without exposing those services to the open internet. This also enables remote administration of the VRB itself (SSH over VPN). By optionally enabling an alternate SSH port on WAN, you have a backup access method if the VPN is unreachable – but day-to-day, VPN is the primary method.

- **Centralized Management with NM:** Using NetworkManager for all networking (even on Ubuntu) simplifies management. You can use `nmcli` or nmtui to adjust connections, add bridges or VLANs, etc. The dnsmasq configs under NetworkManager mean you modify a file and restart NM to update DHCP/DNS for a network. This integrates nicely with a GitOps or config management approach – these config files can be kept in version control (as presumably they are, given the context).

- **Dynamic IP Assignment and Naming:** All client machines and VMs can get IPs dynamically via DHCP while still being reachable by predictable DNS names. For example, if you spin up a new VM and want it on the LAN, you can either give it a static DHCP reservation (so it always gets the same IP and a chosen hostname), or let it take a dynamic IP (which will still get a DNS entry via `dhcp-hostname` if the client provides one). The `expand-hosts` setting means if you register any manual entries in `/etc/hosts` (say for external services or special cases), those will be resolvable by LAN clients too. This network setup is very flexible for a lab or cloud environment where things can change frequently.

- **Firewall and Security:** The deployed firewall rules protect the environment's perimeter. Only the VPN (and selected ports like alternate SSH or game servers if you open them) are accessible from outside, reducing the attack surface. Inside, everything can communicate as needed, which is ideal for a collaborative development environment.

To further integrate with your Spectra ecosystem, you might connect this VRB to other parts of your infrastructure (for example, if there is an existing corporate network, the VRB's LAN could be linked via a VLAN or a static route to that network). Given the modular nature of the config, you can add such routes or VLANs easily. The provided template in the LAN dnsmasq config [4] is a guide for adding a new subnet – just define it there and create the interface (e.g. `eno8303.20` for VLAN20) in NetworkManager, and the VRB will start serving that network as well.

Default Paths Recap: - NetworkManager key config: `/etc/NetworkManager/conf.d/dns.conf` (enables dnsmasq). - DNS/DHCP zone files: `/etc/NetworkManager/dnsmasq.d/` (as detailed above). - Lease files: `/var/lib/NetworkManager/dnsmasq-*.leases` (one per zone, as set in configs). - WireGuard config: `/etc/wireguard/wg0.conf` (and keys in `/etc/wireguard/`). - Script files (if any from this bundle): e.g. `deploy_vrb.sh` and `add_wg_peer.sh` (you can keep them in `/root` or a convenient location).

After deployment, verify each component: 1. **NetworkManager & Dnsmasq:** Run `nmcli dev show` to see IP assignments. Check that `ps aux | grep dnsmasq` shows NetworkManager's dnsmasq running with our config files. Test DNS resolution on the VRB (`dig @127.0.0.1 koalab-router.arquolab.io.net.lan` should return 10.10.10.2). Test from a LAN client as well. 2. **DHCP:** On a LAN client, ensure it receives an IP in range and the DNS is set to 10.10.10.1. Similarly, if you plug a device or assign a VM to the WAN bridge (if configured), see that it can get an IP in the 144.2.68.0/27 range. 3. **WireGuard:** On the VRB, `sudo wg show` to ensure it's listening. From a remote client with the generated config, test connectivity (ping internal hosts, etc.). 4. **Routing/NAT:** From a LAN machine, try accessing an external website – it should work (NAT through VRB). From a VPN client, likewise test internet access. Also test that the LAN machine can reach the VPN client (if needed) and vice versa; by default WireGuard peers don't talk to each other unless AllowedIPs includes each other, but LAN <-> VPN should work one-way: VPN can reach LAN. If you need LAN devices to initiate connections to VPN clients, you'd have to push routes to them or configure the VRB to NAT those as well – usually not needed.

This comprehensive setup ensures a **production-like environment** for game development: multiple network zones, centralized services, secure remote access, and flexibility for expansion. It's tailored to the Spectra Gallery's needs, but you can adapt domain names, IP ranges, and other parameters to fit your exact scenario. By using the provided scripts and configs, the deployment is repeatable and version-controlled, which aligns well with managing a cloud-based development infrastructure.

[1] [5] [6] [7] 03-DHCP-arquolab-io.conf
file://file-QWcRFW2Dy5qj1KAURwPy6n

[2] [4] [8] [9] [10] 02-DHCP-arquolab-io-net-lan.conf
file://file-Lf9rVLXySzTuq7BKTcJWto

[3] 20-DNS-libvirt-lan.conf
file://file-2BswjkRjciNoZvx7JtXDAf

[11] [12] [13] [14] Setup a WireGuard server with client config as QR codes - ./stu.sh
https://stu.sh/posts/setup-wireguard-cli-qr-code/