How to: Get Up and Running with Magic WAN (+ its interops)

Erfi Anugrah

2024-05-24

Table of contents

Step 1: Conduit Configuration

This would be setup during the onboarding with Cloudflare, the setup would require specific information from your end w.r.t specific subnets that should be upgraded or if you want to use non RFC 1918 prefixes.

Step 2: GRE and/or IPsec Tunnel Prerequisites

Both GRE and IPsec would add on top of the raw TCP packet, assuming MTU size being 1500 (this can be lower depending on your internet breakouts, KPN and Deutsch Telecom for instance would already need TCP clamping), the MSS would be lower:

For GRE:

Standard Internet Routable MTU	1500 bytes
- Original IP header	20 bytes
- Original protocol header (TCP)	20 bytes
- New IP header	20 bytes
- New protocol header (GRE)	4 bytes
= Maximum segment size (MSS)	1436 bytes

For IPsec this value would be lower still, depending on if it's IPsec within GRE or on its own. The `ESP` header would be `8 bytes`, `ESP Trailer` could be `16 to 20 bytes` conservatively.

In my case the end value is `1350`.

You can check by running this command:

```
sudo tcpdump -i [INTERFACE] 'tcp[tcpflags] & (tcp-syn)≠0 and dst port [443 or 80]
```

Or running ping with `do-not-fragment` and `size`:

```
sudo ping IP -M -s 1500
```

An example would look like this for the internet breakout:

```
09:47:29.132309 IP [MY_IP].fixed.kpn.net.53958 > 162.159.138.105.https: Flags [S], seq 471050517, win 64240,
↪ options [mss 1452,nop,wscale 8,nop,nop,sackOK], length 0
```

With IPsec, `10.68.100.20` is the tunnel endpoint on my side of the tunnel:

```
09:49:19.268016 IP 10.68.100.20.51678 > 104.16.236.133.https: Flags [S], seq 2995991175, win 32120, options
↪ [mss 1350,sackOK,TS val 1792385600 ecr 0,nop,wscale 7], length 0:
```

Step 3: Configure your tunnels and static routes on Cloudflare

- Create a vti with a /31 subnet for use, refer to your vendor documentation on how to create one
- The Cloudflare endpoint will be provided to you via the conduit configuration yaml
- The Customer endpoint would be the IP provided to you via your ISP
- By default, you can only add static routes with RFC 1918 IP prefixes like:
  - 10.0.0.0/8
  - 172.16.0.0/12
  - 192.168.0.0/16

There are exceptions for publicly routable addresses, inform your friendly (at this point) implementation manager before the project begins.

Step 4: Make sure health-checks work

You should start seeing Cloudflare's side of the tunnel hitting yours (health wise, take an average as not all data centers pinging your end of the tunnel matters):

```
10:05:07.160315 IP 10.68.100.21 > 10.68.100.20: ICMP echo request, id 7295, seq 0, length 64
10:05:07.160378 IP 10.68.100.20 > 10.68.100.21: ICMP echo reply, id 7295, seq 0, length 64
10:05:07.169088 IP 10.68.100.21 > 10.68.100.20: ICMP echo request, id 63043, seq 0, length 64
10:05:07.169150 IP 10.68.100.20 > 10.68.100.21: ICMP echo reply, id 63043, seq 0, length 64
10:05:07.208415 IP 10.68.100.21 > 10.68.100.20: ICMP echo request, id 41057, seq 0, length 64
10:05:07.208498 IP 10.68.100.20 > 10.68.100.21: ICMP echo reply, id 41057, seq 0, length 64
10:05:07.238198 IP 10.68.100.21 > 10.68.100.20: ICMP echo request, id 17771, seq 0, length 64
```

If you have `replay-window` enabled, make sure anti-replay protection is enabled on Cloudflare's side.

Step 5: Route the private subnets to your `vti`

! Important

Check your `sysctl` configuration for linux kernels.
It is possible that you might need to set these values:

```
sudo sysctl -w net.ipv4.conf.all.accept_local=1
sudo sysctl -w net.ipv4.conf.all.rp_filter=0
```

For `rp_filter`:

- 0 - No source validation.
- 1 - Strict mode as defined in RFC3704 Strict Reverse Path Each incoming packet is tested against the FIB and if the interface is not the best reverse path the packet check will fail. By default failed packets are discarded.
- 2 - Loose mode as defined in RFC3704 Loose Reverse Path Each incoming packet's source address is also tested against the FIB and if the source address is not reachable via any interface the packet check will fail.

Current recommended practice in RFC3704 is to enable strict mode to prevent IP spoofing from DDos attacks.
If using asymmetric routing or other complicated routing, then loose mode is recommended.

Consult your vendor documentation, here are some examples:

- Alibaba Cloud VPN Gateway
- Amazon AWS Transit Gateway
- Aruba EdgeConnect Enterprise
- Cisco IOS XE
- Cisco SD-WAN
- Fortinet
- Furukawa Electric FITELnet
- Google Cloud VPN
- Microsoft Azure
- Palo Alto Networks NGFW
- pfSense
- SonicWall
- Sophos Firewall
- strongSwan
- VyOS

You can do a TCP `traceroute` to see if it's going via the tunnel to another endpoint that's also exposed via Magic WAN:

```
sudo traceroute -T 172.18.0.8
traceroute to 172.18.0.8 (172.18.0.8), 30 hops max, 60 byte packets
 1  10.68.69.1 (10.68.69.1)  0.627 ms  0.460 ms  0.480 ms
 2  10.68.100.21 (10.68.100.21)  5.833 ms  5.289 ms  5.264 ms
 3  172.71.93.32 (172.71.93.32)  7.644 ms  7.038 ms  10.645 ms
 4  172.18.0.8 (172.18.0.8)  10.356 ms  10.562 ms  10.244 ms
```

Step 6: Automate your provisioning through the use of Gitops/IaC

This can be done via the UI, API or via Terraform. However, it is recommended to use infrastructure as code as much as possible. The examples make use of `.tfvars` file and a `variables.tf` file to reference those sensitive values when applying or planning the infrastructure with Terraform. There are many ways to go about securing the sensitive information including the `.tfstate` file such as encrypting with `Mozilla SOPS` and `Age`.

GRE Tunnels

```
resource "cloudflare_gre_tunnel" "vyos_sg" {
  account_id              = var.cloudflare_account_id
  name                    = "vyos_sg"
  customer_gre_endpoint   = var.sg_ip
  cloudflare_gre_endpoint = var.wan_ip_1  # This will be provided to you during onboarding
  interface_address       = "10.68.88.21/31"
  description             = "vyos_sg_gre"
  ttl                     = 64
  mtu                     = 1476
  health_check_enabled    = true
  health_check_target     = var.sg_ip
  health_check_type       = "request"
}
```

IPsec Tunnels

```
resource "cloudflare_ipsec_tunnel" "vyos_sg_ipsec" {
  account_id           = var.cloudflare_account_id
  name                 = "vyos_sg_ipsec"
  customer_endpoint    = var.sg_ip
  cloudflare_endpoint  = var.wan_ip_2
  interface_address    = "10.68.77.21/31"
  description          = "vyos_sg_ipsec_m1"
  health_check_enabled = true
  health_check_target  = var.sg_ip
  health_check_type    = "request"
  psk                  = var.psk_sg
  allow_null_cipher    = false
  hex_id               = var.hex_id_sg
  fqdn_id              = var.fqdn_id_sg
  user_id              = var.user_id_sg
}
```

Static Routes

```
The `next-hop` address would be Cloudflare's side of the tunnel
resource "cloudflare_static_route" "eth1_vyos_nl_ipsec" {
  account_id  = var.cloudflare_account_id
  description = "ETH1"
  prefix      = "10.68.69.0/24"
  nexthop     = "10.68.100.20"
  priority    = 100
}
resource "cloudflare_static_route" "eth1_100_vyos_nl_ipsec" {
  account_id  = var.cloudflare_account_id
  description = "VLAN_100"
  prefix      = "10.68.70.0/24"
  nexthop     = "10.68.100.20"
  priority    = 100
}

resource "cloudflare_static_route" "podman_vyos_nl_ipsec" {
  account_id  = var.cloudflare_account_id
  description = "Podman"
  prefix      = "172.18.0.0/16"
  nexthop     = "10.68.100.20"
  priority    = 100
}
```

Magic Firewall

Refer also to example rulesets based on common attack vectors

```hcl
resource "cloudflare_magic_firewall_ruleset" "magic_firewall" {
  account_id  = var.cloudflare_account_id
  name        = "Magic WAN Firewall"
  description = "Default Magic WAN Firewall"

  rules = [
    {
      action      = "allow"
      expression  = "(ip.proto eq \"icmp\")"
      description = "Allow ICMP"
      enabled     = "true"
    },
    {
      action      = "allow"
      expression  = "(ip.src in {100.64.0.0/10})"
      description = "Allow WARP Virtual IPs"
      enabled     = "true"
    }
  ]
}
```

Step 7: Interoperability (Optional)

⚠ Warning

Be aware of these limitations
Virtual Networks

- Ensure Cloudflare Tunnel and WARP are on the default network for the interoperability to function

DH Group for IPsec Tunnels

- Cloudflare supports DH groups 20, 14, 5, but only one should be used when creating tunnels

WARP w/ Magic WAN

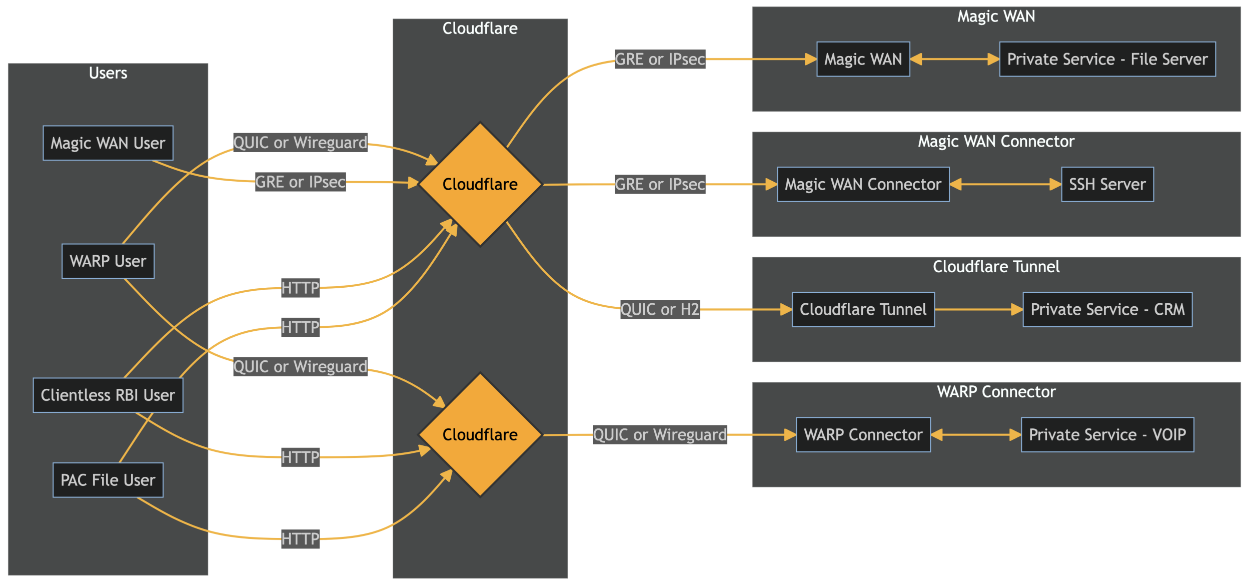
- Ensure WARP connectivity in locations whereby connectivity is routed to Cloudflare Gateway via Magic WAN to be excluded. This is due to the double encapsulation, once by WARP, and again via Magic WAN. Routing policies can be used to exclude connections to the WARP Ingress IPs and WARP UDP Ports

Cloudflare Tunnel w/ Magic WAN

- Cloudflare Tunnel does not support outbound connections. Overlapping routes between Cloudflare Tunnel and Magic WAN will cause issues with outbound connections since Cloudflare Tunnel routes are prioritized over Magic WAN static routes

WARP Connector w/ Magic WAN

- The WARP Connector solves the bi-directional use case that Cloudflare Tunnel doesn't solve, however at this point, does not work with Magic WAN when configured within the same Cloudflare account/organisation (as the conduit configuration is tied to the account), and should be used as an alternative (as there are overlapping use cases) if the traditional approach via Magic WAN does not suit your current infrastructure.



[1]

[1] The arrows signify bidirectional connections or unidirectional

PAC File (Proxy Endpoints)

There are many ways to deploy the PAC file such as MDMs and using a remote server, in this case, I'll be using Workers. You can follow the steps here, when using the code, you can create more cases to match specific files, in my case it is nl.pac and sg.pac to simulate the two locations, I also used Workers Secrets to store the Proxy Endpoint domains:

PAC File Worker Deployment

Worker entrypoint (index.js)

```javascript
import { nl_pac_file, sg_pac_file } from "./pac_file.js";

export default {
  fetch(request, env) {
    const url = new URL(request.url);

    if (url.pathname === "/nl.pac") {
      return nl_pac_file(env);
    } else if (url.pathname === "/sg.pac") {
      return sg_pac_file(env);
    } else {
      return new Response("Not Found", { status: 404 });
    }
  },
};
```

PAC File variables (pac_file.js)

```javascript
export function nl_pac_file(env) {
  const nl = `
function FindProxyForURL(url, host) {
// No proxy for private (RFC 1918) IP addresses (intranet sites)
  // if (
  //   isInNet(dnsResolve(host), "10.0.0.0", "255.0.0.0") ||
  //   isInNet(dnsResolve(host), "172.16.0.0", "255.240.0.0") ||
  //   isInNet(dnsResolve(host), "192.168.0.0", "255.255.0.0")
  // ) {
  //   return "DIRECT";
  // }

  // No proxy for localhost
  // if (isInNet(dnsResolve(host), "127.0.0.0", "255.0.0.0")) {
  //   return "DIRECT";
  // }
  // Example logic to determine whether to use a proxy
  return "HTTPS ${env.NL_DOMAIN}.proxy.cloudflare-gateway.com:443";
}
`;
  // Set headers to prevent caching
  const headers = new Headers({
    "Content-Type": "application/x-ns-proxy-auto-config",
    "Cache-Control": "no-store, max-age=0",
  });

  return new Response(nl, { headers: headers });
}

export function sg_pac_file(env) {
  const sg = `
function FindProxyForURL(url, host) {
// No proxy for private (RFC 1918) IP addresses (intranet sites)
  // if (
  //   isInNet(dnsResolve(host), "10.0.0.0", "255.0.0.0") ||
  //   isInNet(dnsResolve(host), "172.16.0.0", "255.240.0.0") ||
  //   isInNet(dnsResolve(host), "192.168.0.0", "255.255.0.0")
  // ) {
  //   return "DIRECT";
  // }

  // No proxy for localhost
  // if (isInNet(dnsResolve(host), "127.0.0.0", "255.0.0.0")) {
  //   return "DIRECT";
  // }
  // Example logic to determine whether to use a proxy
  return "HTTPS ${env.SG_DOMAIN}.proxy.cloudflare-gateway.com:443";
}
`;
```

```
  // Set headers to prevent caching
  const headers = new Headers({
    "Content-Type": "application/x-ns-proxy-auto-config",
    "Cache-Control": "no-store, max-age=0",
  });

  return new Response(sg, { headers: headers });
}
```

wrangler.toml

```toml
name = "worker-proxy-pac"
main = "src/index.js"
compatibility_date = "2024-05-12"
compatibility_flags = ["nodejs_compat"]

[dev]
port = 9001
local_protocol="http"
upstream_protocol="https"

[env.staging]
name = "staging-pac"
vars = { ENVIRONMENT = "staging" }
workers_dev = true

[env.prod]
name = "prod-pac"
vars = { ENVIRONMENT = "production" }
routes = [
    { pattern = "proxy.{DOMAIN}.com", custom_domain = true },
]
# Secrets
# NL_DOMAIN
# SG_DOMAIN
```

Gitops/IaC

Proxy Endpoints HCL

```hcl
resource "cloudflare_teams_proxy_endpoint" "nl_proxy_endpoint" {
  account_id = var.cloudflare_account_id
  name       = "nl"
  ips        = ["${var.nl_ip}/32"]
}

resource "cloudflare_teams_proxy_endpoint" "sg_proxy_endpoint" {
  account_id = var.cloudflare_account_id
  name       = "sg"
  ips        = ["${var.sg_ip}/32"]
}
```

WARP/Clientless RBI

With WARP, the managed deployment approach is recommended as you can ensure that not only you're managing the devices and the policies but also the WARP client itself. This can also be managed through the use of Device Profiles. If the use case is to connect to endpoints behind Cloudflare Tunnel or Magic WAN, select the `default` Virtual Network in the client's dropdown menu.

Both WARP (you can also isolate applications) and Clientless RBI would use the same Access application and policy as shown below, and both would allow the user to access private IP applications via the other on-ramps.

Gitops/Iac

WARP/RBI Access Policy HCL

```hcl
resource "cloudflare_access_policy" "warp_login" {
  account_id       = var.cloudflare_account_id
  name             = "Allow Erfi"
  decision         = "allow"
  session_duration = "30m"

  include {
    group = [cloudflare_access_group.erfi_corp.id]
  }
}
```

WARP/RBI Access App HCL

```hcl
resource "cloudflare_access_application" "warp_login" {
  account_id = var.cloudflare_account_id
  policies = [
    cloudflare_access_policy.warp_login.id
  ]
  allowed_idps = [
    cloudflare_access_identity_provider.entra_id.id,
    cloudflare_access_identity_provider.google_workspace.id,
    cloudflare_access_identity_provider.gmail.id,
    cloudflare_access_identity_provider.keycloak_oidc.id,
    cloudflare_access_identity_provider.authentik_oidc.id,
    cloudflare_access_identity_provider.authentik_saml.id,
    cloudflare_access_identity_provider.otp.id
  ]
  auto_redirect_to_identity = false
  domain                    = "erfianugrah.cloudflareaccess.com/warp"
  name                      = "Warp Login App"
  session_duration          = "24h"
  type                      = "warp"
}
```

Virtual Networks HCL

```hcl
resource "cloudflare_tunnel_virtual_network" "vyos_nl" {
  account_id = var.cloudflare_account_id
  name       = "vyos_nl_vnet"
  is_default_network = true
}
```

Device Settings Policy HCL

```hcl
resource "cloudflare_device_settings_policy" "default" {
  account_id            = var.cloudflare_account_id
  name                  = "default"
  description           = "default_policy"
  # precedence            = 100
  # match                 = "any(identity.groups.name[*] in {\"Erfi Corp\"})"
  default               = true
  enabled               = true
  allow_mode_switch     = true
  allow_updates         = true
  allowed_to_leave      = true
  auto_connect          = 0
  captive_portal        = 180
  disable_auto_fallback = false
  switch_locked         = false
  service_mode_v2_mode  = "warp"
  service_mode_v2_port  = 3000
  exclude_office_ips    = true
}

resource "cloudflare_device_settings_policy" "google" {
  account_id            = var.cloudflare_account_id
  name                  = "Google Workspace"
  description           = "google_workspace_policy"
  precedence            = 200
  match                 = "any(identity.groups.name[*] in {\"Erfi Corp\"})"
  default               = false
  enabled               = true
  allow_mode_switch     = true
  allow_updates         = true
  allowed_to_leave      = true
  auto_connect          = 0
  captive_portal        = 180
  disable_auto_fallback = false
  switch_locked         = false
  service_mode_v2_mode  = "warp"
  service_mode_v2_port  = 3000
  exclude_office_ips    = true
}
```

Cloudflare Tunnel

Tunnels are used to exposed private applications or for connectivity in the case of RDP, SSH, VNC and the like, it does not support bi-directional traffic as mentioned above. By setting up the routes, you can now reach those same private applications behind Cloudflare Tunnel be it from a PAC file, clientless RBI, WARP or behind Magic WAN.

Note

Cloudflare Tunnel Deployment
There are (again) like WARP many ways to deploy, as systemd on a VM or metal, docker standalone or in compose or a deployment in k8s or docker swarm.
Be aware of the system requirements:
For most use cases, we recommend the following baseline configuration:

- Run a cloudflared replica on two dedicated host machines per network location. Using two hosts enables server-side redundancy and traffic balancing.
- Size each host with minimum 4GB of RAM and 4 CPU cores.
- Allocate 50,000 ports to the cloudflared process on each host.

This setup is usually sufficient to handle traffic from 8,000 WARP users (4,000 per host). The actual amount of resources used by cloudflared will depend on many variables, including the number of requests per second, bandwidth, network path and hardware. As additional users are onboarded, or if network traffic increases beyond your existing tunnel capacity, you can scale your tunnel by adding an additional cloudflared host in that location.

Gitops/IaC

Terraform provider HCL with random provider

```
terraform {
  required_providers {
    cloudflare = {
      source  = "cloudflare/cloudflare"
      version = "~> 4.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~> 3.0"
    }
  }
}
```

Use the random provider to generate string that will be use to set the tunnel secret

```
# Generate a random string
resource "random_string" "tunnel_secret" {
  length  = 32
  special = false
}
```

Use the random string as the tunnel secret to create the Cloudflare Tunnel

```
resource "cloudflare_tunnel" "vyos_nl" {
  account_id = var.cloudflare_account_id
  name       = "vyos_nl"
  secret     = base64encode(random_string.tunnel_secret.result)
  config_src = "cloudflare"
}
```

Warning

Again, be aware of conflicting routes between Cloudflare Tunnel and Magic WAN

Tunnel Route HCL

```hcl
resource "cloudflare_tunnel_route" "vyos_nl" {
  account_id         = var.cloudflare_account_id
  tunnel_id          = cloudflare_tunnel.vyos_nl.id
  network            = "0.0.0.0/0"
  virtual_network_id = cloudflare_tunnel_virtual_network.vyos_nl.id
}
```

Tunnel Config HCL

```hcl
resource "cloudflare_tunnel_config" "vyos_nl" {
  account_id = var.cloudflare_account_id
  tunnel_id  = cloudflare_tunnel.vyos_nl.id

  config {
    warp_routing {
      enabled = true
    }
    ingress_rule {
      hostname = "prom-tunnel-nl.${var.domain_name}"
      service  = "http://localhost:11000"
    }
    ingress_rule {
      hostname = "prom-caddy-nl.${var.domain_name}"
      service  = "http://172.18.0.4:2018"
    }
    ingress_rule {
      service = "http_status:404"
    }
  }
}
```

(Beta) WARP Connector

It shares the same functionality as Cloudflare Tunnel does but with bidirectional support. This means that only you can expose private services/applications which require two-way traffic such as communication services.

As the WARP connector is also essentially a device connected via WARP to the internet, a remote user with WARP connectivity can also connect to it directly and not just the services exposed behind it which is what we call WARP-to-WARP connectivity. Both site-to-site and peer-to-peer use cases work here, similar to Magic WAN.

Site-to-site Connectivity

1. Setup Access policy for WARP enrolment (this would be the same policy for WARP authentication), since it's site-to-site, we will have to deploy it via `.xml` with the required parameters.

Refer to service tokens and device enrolment for details.

2. Setup the `split tunnel` to either `include` or `exclude` mode, in this example, I have chosen `exclude` so that I can still SSH into the nodes to showcase the functionality.

Since this portion can be managed by Terraform, this is the example HCL:

```
resource "cloudflare_split_tunnel" "default_include" {
  account_id = var.cloudflare_account_id
  policy_id  = cloudflare_device_settings_policy.default.id
  mode       = "exclude"
  tunnels {
    address     = "10.68.69.3"
    description = "ERFI1"
  }
  tunnels {
    address     = "10.68.73.3"
    description = "arch-0"
  }
  tunnels {
    address     = "10.68.73.2"
    description = "pve"
  }
}
```

3. Ensure these settings are enabled:

- Enable CGNAT routing (Settings -> Network)
  - Enable Proxy (UDP, TCP, ICMP)
  - Enable WARP to WARP
  - Override local interface IP (Settings -> WARP Client)
    - If in `include` mode, ensure `100.96.0.0/12` is included

Some of these settings are available as a Terraform resource (as shown below), since it would be a mix a of API/UI and Terraform, Terraform would not know the state, so just keep to API/UI for all settings.

```
resource "cloudflare_teams_account" "miau3" {
  account_id                             = var.cloudflare_account_id
  tls_decrypt_enabled                    = false
  protocol_detection_enabled             = false
  activity_log_enabled                   = true
  non_identity_browser_isolation_enabled = true

  body_scanning {
    inspection_mode = "deep"
  }

  antivirus {
    enabled_download_phase = false
    enabled_upload_phase   = false
    fail_closed            = false
  }

  fips {
    tls = false
  }

  proxy {
    tcp     = true
    udp     = true
    root_ca = true
  }

  url_browser_isolation_enabled = true

  logging {
    redact_pii = false
    settings_by_rule_type {
      dns {
        log_all    = true
        log_blocks = false
      }
      http {
        log_all    = true
        log_blocks = false
      }
      l4 {
        log_all     = true
        log_blocks = false
      }
    }
  }

  extended_email_matching {
    enabled = true
  }
}
```

4. Set up the WARP Connector

- Either via API or the UI, create the WARP connector (currently not available as a Terraform resource):

```
curl --request POST \
  --url https://api.cloudflare.com/client/v4/accounts/account_id/warp_connector \
  --header 'Content-Type: application/json' \
  --header 'X-Auth-Email: ' \
  --data '{
  "name": "arch-0"
}'
```

Upon creation, you will get the example `.xml` e.g. which will look like this:

```
<dict>
 <key>organization</key>
 <string>org_name</string>
 <key>auth_client_id</key>
 <string>client_access_id_string</string>
 <key>auth_client_secret</key>
 <string>client_access_secret_string</string>
 <key>warp_connector_token</key>
 <string>connector_token_string</string>
</dict>
```

This is what you will place in the `/var/lib/cloudflare-warp` directory after the WARP client install. Refer to the relevant repositories that matches your Linux distribution, for Arch, you would have to install via AUR.

Once done, if not yet enabled, run `sudo systemctl enable warp-svc` then `sudo systemctl start warp-svc`, if already running as a service, run `sudo systemctl restart warp-svc` so that it will use the new configuration to connect to Cloudflare.

5. Set up Linux networking

We need to enable IP forwarding and MSS clamping (WARP's MTU is 1280 bytes).

Create a `.conf` file in the `etc/sysctl.d/` directory with value of `net.ipv4.ip_forward = 1` so that this persists upon reboot.

To check what IPtables rules are existing, we can run `sudo iptables-save > test.conf` to just see what's currently being use, if all's well. We can add the following to the existing `/etc/iptables/iptables.rules` file, such that it would look something like this:

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A FORWARD -i CloudflareWARP -p tcp -m tcp --tcp-flags SYN,RST SYN -j TCPMSS --clamp-mss-to-pmtu
-A FORWARD -o CloudflareWARP -p tcp -m tcp --tcp-flags SYN,RST SYN -j TCPMSS --clamp-mss-to-pmtu
COMMIT
```

This would make sure the MSS clamping rules will also persist on reboot.

6. Route nodes/VMs/containers to the WARP connector

The most common use case here would either be an alternate gateway to the main router or an intermediate gateway. In this example, I will be using it as an intermediate gateway.

Set up the `static routes` for each WARP connector, the configuration will be similar to the `Cloudflare Tunnel` routes. In our case, on the WARP connector called `arch-1`, we will exposing `10.68.73.102/32` for the `arch-2` machine, and `arch-3` will be exposing `10.68.73.104/32` for `arch-4` machine.

Let's use `arch-2` as an example as the same steps will be replicated on the other node.

Run:

```
ip route
```

and you should be able to see default routes, for instance:

```
default via 10.68.73.1 dev ens18 proto dhcp src 10.68.73.102 metric 100
```

we want to create another route that has higher precedence than this (lower number).

Run:

```
sudo ip route add default via 10.68.73.101 dev ens18 metric 99
```

What this means is that, traffic from `arch-2` will go via the `ens18` interface to `10.68.73.101` which is the WARP connector `arch-1` and using it as a gateway, now this node is connected to Cloudflare.

Do the same for the other node, and change the IP.

Run:

```
curl https://icanhazip.com
```

on either machine, and you should be able to see a Cloudflare IP, just be aware since all traffic is sent to the WARP connector, that the DNS server used by the machines routed to WARP connector can be accessed while on the network, either by another route or a public DNS server.

```
curl icanhazip.com
104.28.218.39
```

7. Route the machines to each other

We can either router the entire prefix to the WARP connector or specific IPs, in my case, I will be using just one IP:

```
sudo ip route add 10.68.73.104 via 10.68.73.101 dev ens18 metric 98
```

This will allow `10.68.73.102` to reach `10.68.73.104`, do the same on `arch-4`, just by swapping the IPs, from `.104` to `.102`

Run:

```
ping 10.68.73.102
```

or

```
mtr 10.68.73.102
```

and vice versa to see the traffic reach the WARP connector CGNAT IP before reaching the other machine.