SASE Gateway GRE/TGW Integration Automation

Discussion Document

Date: 2026-01-27 **Status:** Draft - For Review **Author:** Infrastructure Team

---

1. Executive Summary

This document evaluates options for automating the configuration of eBGP over GRE tunnels between Netskope SASE Gateways and Third parties. The current Netskope BWAN API (v1/v2) does not expose GRE interface configuration, requiring alternative automation approaches.

---

2. Background

2.1 Configuration Requirements

Step	Component	Method	API Support
1	Create SASE Gateway	Terraform Provider	▯ Yes
2	Configure GE2 Interface	Terraform Provider	▯ Yes
3	**Create GRE Tunnel**	**CLI on Gateway**	▯ **No**
4	Configure BGP Peer	Terraform Provider	▯ Yes
5	**Enable default-originate**	**FRR config file**	▯ **No**
6	Create TGW Connect Attachment	AWS Terraform	▯ Yes
7	Create TGW Connect Peer	AWS Terraform	▯ Yes

2.2 CLI Commands Required on Gateway

GRE Tunnel Creation:

```
 infhostd config-gre \
   -inside-ip 169.254.101.1 \
   -inside-mask 255.255.255.248 \
   -intfname gre1 \
   -local-ip 192.168.100.25 \
   -remote-ip 192.0.1.15 \
   -mtu 1300 \
   -phy-intfname ens6

service infhost restart
infhostd restart-container
```

FRR Configuration for Default Route Advertisement:

```
cat > /infroot/workdir/frrcmds-user.json << 'EOF'
{
  "frrCmdSets": [
    {
      "frrCmds": [
        "conf t",
        "router bgp 400",
        "neighbor 169.254.101.2 disable-connected-check",
        "neighbor 169.254.101.2 ebgp-multihop 2",
        "neighbor 169.254.101.3 disable-connected-check",
        "neighbor 169.254.101.3 ebgp-multihop 2",
        "address-family ipv4 unicast",
        "neighbor 169.254.101.2 default-originate",
        "neighbor 169.254.101.3 default-originate"
      ]
    }
  ]
}
EOF

infhostd restart-container
```

3. Automation Options

3.1 Option A: AWS Systems Manager (SSM) Run Command

Description: Use AWS SSM Run Command to execute commands on the SASE Gateway EC2 instance without requiring SSH access.

Architecture:

```
Terraform ──▶ aws_ssm_document ──▶ SSM Service ──▶ SSM Agent ──▶ SASE Gateway
                                        │
                                  CloudTrail
                                  (Audit Log)
```

Terraform Implementation:

```
resource "aws_ssm_document" "configure_sase_gre" {
  name          = "ConfigureSASEGatewayGRE-${var.environment}"
  document_type = "Command"

  content = jsonencode({
    schemaVersion = "2.2"
    description   = "Configure GRE tunnel and FRR on SASE Gateway for TGW integration"
    parameters = {
      greInsideIp   = { type = "String", description = "GRE Inside IP (e.g., 169.254.101.1)" }
      greInsideMask = { type = "String", default = "255.255.255.248" }
      greIntfName   = { type = "String", default = "gre1" }
      greLocalIp    = { type = "String", description = "Local GRE Outside IP (GE2 private IP)" }
      greRemoteIp   = { type = "String", description = "Remote GRE Outside IP (TGW GRE address)" }
      greMtu        = { type = "String", default = "1300" }
      phyIntfName   = { type = "String", default = "ens6" }
      bgpAsn        = { type = "String", default = "400" }
      tgwBgpPeer1   = { type = "String", description = "TGW BGP Peer 1 IP" }
      tgwBgpPeer2   = { type = "String", description = "TGW BGP Peer 2 IP" }
    }
    mainSteps = [
      {
        action = "aws:runShellScript"
        name   = "configureGRETunnel"
        inputs = {
          runCommand = [
            "#!/bin/bash",
            "set -e",
            "echo 'Configuring GRE tunnel...'",
            "infhostd config-gre -inside-ip {{greInsideIp}} -inside-mask {{greInsideMask}} -intfname {{greIntfName}} -local-ip {{gre
            "echo 'Restarting infhost service...'",
            "service infhost restart",
            "sleep 10"
          ]
        }
      },
      {
        action = "aws:runShellScript"
        name   = "configureFRRDefaultOriginate"
        inputs = {
          runCommand = [
            "#!/bin/bash",
            "set -e",
            "echo 'Configuring FRR for default-originate...'",
            "cat > /infroot/workdir/frrcmds-user.json << 'FRREOF'",
            "{",
            "  \"frrCmdSets\": [",
            "    {",
            "      \"frrCmds\": [",
            "        \"conf t\",",
            "        \"router bgp {{bgpAsn}}\",",
            "        \"neighbor {{tgwBgpPeer1}} disable-connected-check\",",
            "        \"neighbor {{tgwBgpPeer1}} ebgp-multihop 2\",",
            "        \"neighbor {{tgwBgpPeer2}} disable-connected-check\",",
            "        \"neighbor {{tgwBgpPeer2}} ebgp-multihop 2\",",
            "        \"address-family ipv4 unicast\",",
            "        \"neighbor {{tgwBgpPeer1}} default-originate\",",
            "        \"neighbor {{tgwBgpPeer2}} default-originate\"",
            "      ]",
            "    }",
            "  ]",
            "}",
            "FRREOF",
            "echo 'Restarting container...'",
            "infhostd restart-container",
            "echo 'Configuration complete.'"
          ]
        }
      }
    ]
  })
}

resource "null_resource" "execute_gre_config" {
  depends_on = [
    aws_instance.sase_gateway,
    aws_ec2_transit_gateway_connect.sase
  ]

  triggers = {
    gre_config = sha256(jsonencode({
      inside_ip = var.gre_inside_ip
      remote_ip = var.tgw_gre_address
    }))
  }

  provisioner "local-exec" {
    command = <<-EOT
      aws ssm send-command \
        --document-name "${aws_ssm_document.configure_sase_gre.name}" \
        --instance-ids "${aws_instance.sase_gateway.id}" \
        --parameters 'greInsideIp=${var.gre_inside_ip},greLocalIp=${aws_instance.sase_gateway.private_ip},greRemoteIp=${var.tgw_gre_
      --region ${var.aws_region}
    EOT
  }
}
```

Pros:

- Native AWS service, no additional infrastructure
- No SSH keys to manage
- Full audit trail via CloudTrail
- Works through NAT (no public IP required on gateway)
- Can be triggered post-deployment for updates

Cons:

- Requires SSM Agent on gateway AMI (⚠ UNKNOWN - see Section 4)
- Requires IAM instance profile with SSM permissions
- Requires VPC endpoint or internet access for SSM service

Effort Estimate: Medium

3.2 Option B: Terraform SSH Provisioner

Description: Use Terraform's `remote-exec` provisioner to SSH directly into the gateway and execute commands.

Terraform Implementation:

```
resource "null_resource" "configure_gre_ssh" {
  depends_on = [
    aws_instance.sase_gateway,
    aws_ec2_transit_gateway_connect.sase
  ]

  connection {
    type        = "ssh"
    user        = "root"  # or appropriate user
    private_key = file(var.ssh_private_key_path)
    host        = var.use_public_ip ? aws_instance.sase_gateway.public_ip : aws_instance.sase_gateway.private_ip

    # If using bastion
    bastion_host        = var.bastion_host
    bastion_user        = var.bastion_user
    bastion_private_key = file(var.bastion_key_path)
  }

  provisioner "file" {
    content = templatefile("${path.module}/templates/frrcmds-user.json.tpl", {
      bgp_asn      = var.bgp_asn
      tgw_bgp_peer_1 = var.tgw_bgp_peer_1
      tgw_bgp_peer_2 = var.tgw_bgp_peer_2
    })
    destination = "/infroot/workdir/frrcmds-user.json"
  }

  provisioner "remote-exec" {
    inline = [
      "infhostd config-gre -inside-ip ${var.gre_inside_ip} -inside-mask 255.255.255.248 -intfname gre1 -local-ip ${aws_instance.sase
      "service infhost restart",
      "sleep 10",
      "infhostd restart-container"
    ]
  }
}
```

Pros:

- Simple to implement
- No AWS service dependencies

- Works with any Linux-based gateway

Cons:

- Requires SSH access (security group rules, key management)
- SSH keys must be securely managed
- Not idempotent by default
- Requires public IP or bastion host
- No built-in audit trail

Effort Estimate: Low

3.3 Option C: EC2 User Data (Launch-Time Only)

Description: Include GRE configuration in the EC2 instance user data script, executed at first boot.

Terraform Implementation:

```
resource "aws_instance" "sase_gateway" {
  ami           = var.sase_gateway_ami
  instance_type = var.instance_type

  user_data = base64encode(templatefile("${path.module}/templates/userdata.sh.tpl", {
    gre_inside_ip    = var.gre_inside_ip
    gre_local_ip     = "SELF"  # Will be determined at boot
    gre_remote_ip    = var.tgw_gre_address
    bgp_asn          = var.bgp_asn
    tgw_bgp_peer_1   = var.tgw_bgp_peer_1
    tgw_bgp_peer_2   = var.tgw_bgp_peer_2
  }))

  # ... other configuration
}
```

userdata.sh.tpl:

```
 #!/bin/bash
# Wait for network and services
sleep 60

# Get local IP from metadata
LOCAL_IP=$(curl -s http://169.254.169.254/latest/meta-data/local-ipv4)

# Configure GRE
infhostd config-gre \
  -inside-ip ${gre_inside_ip} \
  -inside-mask 255.255.255.248 \
  -intfname gre1 \
  -local-ip $LOCAL_IP \
  -remote-ip ${gre_remote_ip} \
  -mtu 1300 \
  -phy-intfname ens6

service infhost restart
sleep 10

# Configure FRR
cat > /infroot/workdir/frrcmds-user.json << 'EOF'
{
  "frrCmdSets": [{
    "frrCmds": [
      "conf t",
      "router bgp ${bgp_asn}",
      "neighbor ${tgw_bgp_peer_1} disable-connected-check",
      "neighbor ${tgw_bgp_peer_1} ebgp-multihop 2",
      "neighbor ${tgw_bgp_peer_2} disable-connected-check",
      "neighbor ${tgw_bgp_peer_2} ebgp-multihop 2",
      "address-family ipv4 unicast",
      "neighbor ${tgw_bgp_peer_1} default-originate",
      "neighbor ${tgw_bgp_peer_2} default-originate"
    ]
  }]
}
EOF

infhostd restart-container
```

Pros:

- No external access required
- Runs automatically at instance launch
- Simple implementation

Cons:

- Only runs at initial deployment
- Cannot update configuration without instance replacement
- TGW must be created before gateway (chicken-and-egg problem)
- Debugging issues is difficult

Effort Estimate: Low

3.4 Option D: AWS Lambda + SSM (Event-Driven)

Description: Create a Lambda function that configures the gateway via SSM, triggered by CloudWatch Events or Step Functions.

Architecture:

```
Terraform ──▶ Lambda ──▶ SSM ──▶ Gateway
     │            │
     │      Step Functions
     │       (orchestration)
     │
     └──▶ EventBridge (trigger on TGW ready)
```

Pros:

- Event-driven, can react to infrastructure changes
- Can include retry logic and error handling
- Can be part of larger orchestration workflow
- Good for complex multi-step deployments

Cons:

- Most complex to implement
- Additional Lambda function to maintain
- Still requires SSM agent on gateway

Effort Estimate: High

4. Unknown Items Requiring Investigation

4.1 CRITICAL: SSM Agent on SASE Gateway AMI

Question	Status	Impact
Is SSM Agent pre-installed on Netskope SASE Gateway AMI?	⚠ **UNKNOWN**	Blocks Option A, D
If not, can it be installed via user data at launch?	⚠ **UNKNOWN**	Mitigation for above
What is the default OS user for SSH access?	⚠ **UNKNOWN**	Impacts Option B

Action Required:

1. Launch a test SASE Gateway instance
2. Check for SSM agent: `systemctl status amazon-ssm-agent`
3. If not present, test installation: `yum install -y amazon-ssm-agent` or `snap install amazon-ssm-agent`

4.2 Configuration Persistence

Question	Status	Impact
Does GRE config survive instance reboot?	⚠ **UNKNOWN**	May need to re-run config
Does GRE config survive `infhostd` upgrades?	⚠ **UNKNOWN**	Upgrade procedures
Is `/infroot/workdir/` persistent storage?	⚠ **UNKNOWN**	FRR config persistence

4.5 IAM and Security

Question	Status	Impact
What IAM permissions does the gateway instance need for SSM?	Documented	`AmazonSSMManagedInstanceCore`
Are there VPC endpoints available for SSM in the deployment region?	Check per region	Option A network path
What is the root password / SSH key for the gateway?	⚠ **UNKNOWN**	Option B access

5. Comparison Matrix

Criteria	Option A (SSM)	Option B (SSH)	Option C (UserData)	Option D (Lambda)
Implementation Effort	Medium	Low	Low	High
Security	□ High	⚠ Medium	□ High	□ High
Audit Trail	□ CloudTrail	□ None	□ None	□ CloudWatch
Idempotent	□ Yes	⚠ Manual	□ No	□ Yes
Post-Deploy Updates	□ Yes	□ Yes	□ No	□ Yes
Requires SSM Agent	□ Yes	□ No	□ No	□ Yes
Requires SSH Access	□ No	□ Yes	□ No	□ No
Works in Private Subnet	□ Yes*	⚠ Bastion	□ Yes	□ Yes*
Maintenance Overhead	Low	Low	Low	Medium

*Requires VPC endpoints for SSM

6. Recommended Approach

Primary Recommendation: Option A (SSM) with Option C (UserData) Fallback

Rationale:

1. SSM provides the best balance of security, auditability, and flexibility
2. User data provides a fallback for initial configuration if SSM is not available
3. Both can coexist - user data for initial setup, SSM for updates

Implementation Phases:

Phase 1: Investigation (1-2 days)

- ☐ Verify SSM agent availability on SASE Gateway AMI
- ☐ Confirm configuration persistence

Phase 2: Prototype (2-3 days)

- ☐ Create SSM document for GRE configuration
- ☐ Create Terraform module combining:
    - netskopebwan provider for gateway/BGP
    - AWS provider for TGW/SSM
- ☐ Test end-to-end deployment

Phase 3: Eval Ready (6-8 days)

- ☐ Add error handling and retry logic
- ☐ Create monitoring/alerting for configuration drift
- ☐ Document runbooks for troubleshooting
- ☐ Create automated tests

Alternative: Option B (SSH) if SSM Not Available

If SSM agent is not available and cannot be installed:

1. Use SSH provisioner with bastion host
2. Implement proper key management (Secrets Manager or Vault)
3. Add security group rules for SSH from Terraform runner