SCENARIO 04 · GWLB GENEVE INSPECTION · DEPLOYMENT-KEY

Transparent inspection with an AWS Gateway Load Balancer

A horizontally-scaled, highly-available inspection architecture. Workload traffic is steered to a GWLB endpoint that GENEVE-encapsulates each flow to a fleet of Enforza engines — one per AZ. Each engine decapsulates, applies policy, source-NATs to the internet, and returns replies through the tunnel. Deploy with Terraform or CloudFormation.

◆ Deployment-key variant

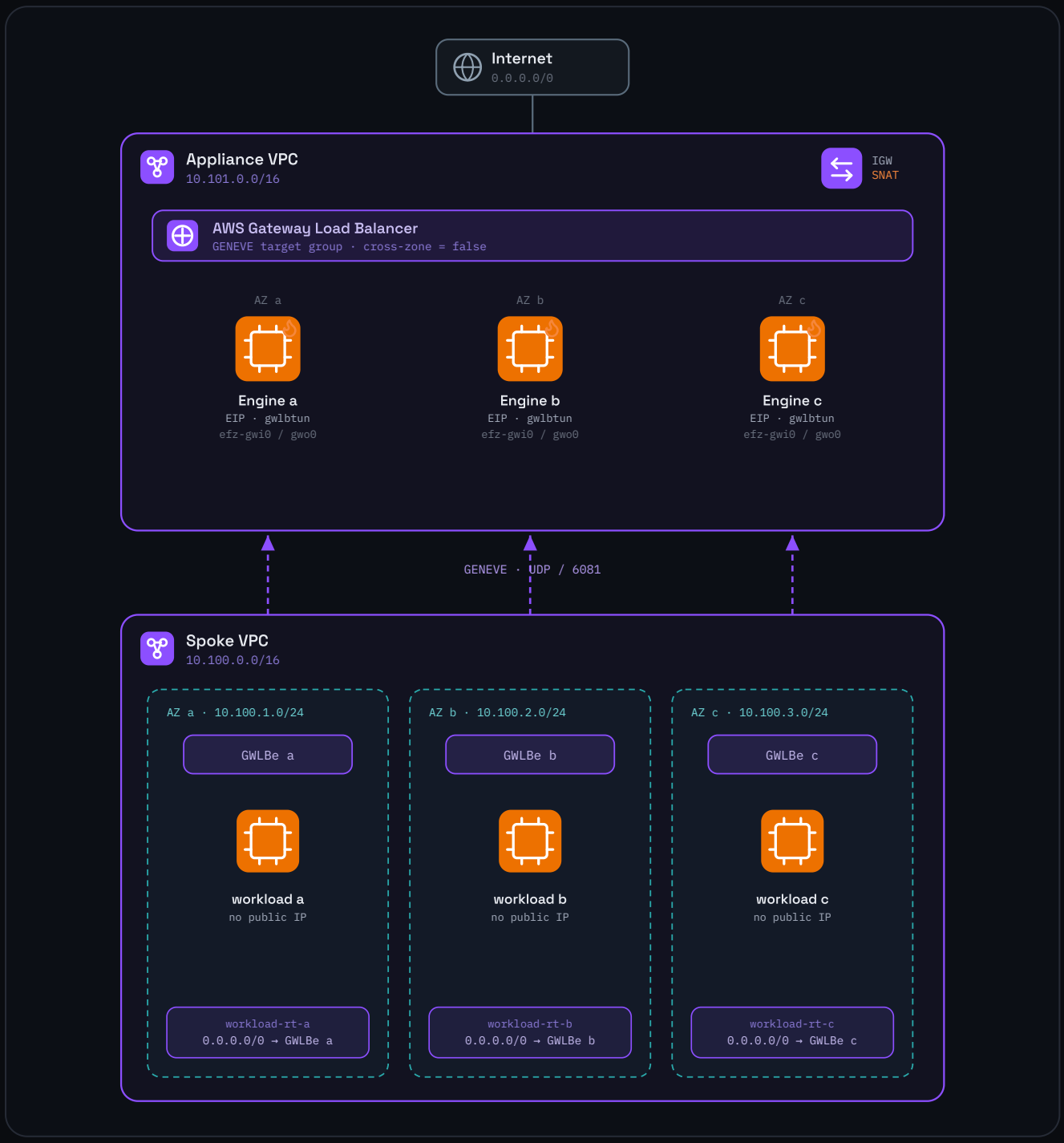
Engines launch from a stock Debian 12 base AMI and bootstrap from the public CDN at first boot with a fleet deployment key. No AWS Marketplace subscription. Terraform and CloudFormation build the identical 3-AZ topology, CIDRs and outputs.

YOU'LL NEED

- A fleet (multi-use) deployment key — one key registers all three engines (see page 4).
- An EC2 key pair, and a region with three AZs.
- `Terraform ≥ 1.5` (AWS provider ~> 5.0) or the AWS CLI, with credentials configured.

The architecture

Two VPCs. The spoke VPC holds workloads whose default route points at an AZ-local GWLB endpoint; the appliance VPC holds the GWLB and one engine per AZ. Flows are GENEVE-tunnelled to the engine in the same AZ — cross-zone off keeps each AZ's traffic in-AZ.



Networking — VPC · GWLB · GWLBe · route Compute — engine & workload EC2

AZ workload subnet

When to use GWLB, and what it deploys

Transparent inspection	Workloads keep egressing to a logical endpoint — no per-instance route to a firewall ENI.
Horizontal scale + HA	N engines in a GWLB target group; GWLB load-balances flows and removes unhealthy engines automatically.
Multi-AZ affinity	With cross-zone off, each AZ's traffic is inspected by that AZ's engine — no cross-AZ data transfer, AZ-isolated blast radius.

◆ **Single appliance instead?**

If you only need one appliance or simple route-through NAT, prefer the single-gateway scenario. GWLB suits scale-out inspection and per-AZ isolation.

WHAT IT DEPLOYS

- Appliance VPC (`10.101.0.0/16`) with an IGW and one subnet per AZ.
- One Enforza engine per AZ (3 total) on a Debian 12 AMI, each with an Elastic IP, source/dest check off, registered behind the GWLB.
- An AWS Gateway Load Balancer + GENEVE target group + a VPC endpoint service (`cross_zone_lb = false` for AZ affinity).
- Spoke VPC (`10.100.0.0/16`) with an IGW and, in each AZ, a workload subnet, a GWLB endpoint, and a sample workload whose default route points at its AZ-local GWLBe.
- Optional `management_cidrs` → `/32` bypass routes so you can SSH the workloads directly.

All resources tagged `Project=enforza` and `Scenario=04-gwlb-deployment-key`.

PREREQUISITE

Get a fleet deployment key

1 Generate a multi-use key in the console

Console (CCX): sidebar → Onboard Firewall → Deployment Keys →

- Generate a fleet (multi-use) key — not single-use
- Copy it; it looks like `efz_xxxxxxxxxxxxxxx`
- You'll pass the same key to all three engines via the deploy command

EXPLANATION — GWLB registers three engines from one deploy. A fleet key can register all of them; a one-time/per-firewall key would only register the first engine to boot and the others would fail with "invalid or expired key". This is the same Deployment Keys screen used in the AWS Lab and single-gateway guides — just choose the multi-use option.

△ Fleet key, not single-use

The one thing that differs from the single-gateway deploy: here you must use a multi-use key so all three AZ engines can register with it.

HOW IT COMES UP

The first-boot bootstrap

1. **Engines boot & bootstrap.** Each engine boots from the stock Debian 12 AMI; cloud-init fetches the public installer and passes it the fleet key.
2. **Engine + sidecar install.** The installer sets up the Enforza engine and its `enforza-gwlbtun` sidecar, and registers with the Enforza cloud using the fleet key.
3. **Enable the GWLB connector — on each engine.** The engines appear in the console. On each firewall, enable the AWS GWLB connector. That starts the sidecar, brings up the `efz-gwi0`/`efz-gwo0` tunnels, and programs the GENEVE datapath.
4. **Health checks go green.** GWLB health checks (TCP, default port 9092) pass and GWLB starts forwarding flows to the engines.

> △ **GWLB starts disabled — this is deliberate**
>
> The connector is off on a fresh engine, so nothing tunnels until you flip the per-firewall toggle in the console. Do it on all three engines. Until then, GWLB health checks stay unhealthy and no flows are forwarded.

> **EXPLANATION —** The installer targets prod (`https://dl.neon.efz.io/install.sh`). The GWLB datapath requires Debian 12 (bookworm) or Ubuntu 22.04+ — `base_ami_id` defaults to official Debian 12 in `eu-west-2`. In other regions, pass your own `base_ami_id`.

OPTION A · TERRAFORM

Deploy with Terraform

2 Init and apply

Work in the `terraform/` directory and pass your fleet key, key pair and a resource prefix.

bash — terraform

```
$ cd terraform
$ terraform init
$ terraform apply \
-var deployment_key=efz_xxxxxxxxxxxxxxxx \
-var ssh_key_name=my-key \
-var resource_prefix=acme-gwlb
```

EXPLANATION — This builds the full 3-AZ topology — both VPCs, the GWLB and endpoint service, three engines and three workloads. Optional overrides: `-var base_ami_id=ami-xxxx` (other regions / pre-baked AMI), `-var instance_type=c6i.xlarge`, `-var management_cidrs='["203.0.113.4/32"]'` (SSH bypass to the sample workloads), and the CIDR vars (`appliance_vpc_cidr`, `spoke_vpc_cidr`).

✓ Then enable the connector

After apply, enable the GWLB connector on each engine in the console (bootstrap step 3). Nothing inspects until all three are on.

OPTION B · CLOUDFORMATION

Deploy with CloudFormation

2 Deploy the stack

The template builds the identical 3-AZ topology. It creates no IAM resources, so no `--capabilities` flag is needed.

bash — aws-cli

```
$ aws cloudformation deploy \
--template-file cloudformation/template.yaml \
--stack-name enforza-gwlb-deployment-key \
--parameter-overrides \
        DeploymentKey=efz_xxxxxxxxxxxxxxx \
        SshKeyName=my-key \
        ResourcePrefix=acme-gwlb
```

EXPLANATION — Prefer clicking? Upload `template.yaml` via CloudFormation → Create stack → Upload a template file, then fill the same parameters in the form (no IAM acknowledgement needed here). Optional keys mirror the Terraform vars: `BaseAmiId`, `InstanceType`, `ApplianceVpcCidr`, `SpokeVpcCidr`, and `ManagementCidrs=203.0.113.4/32` (SSH bypass; the first entry is wired as a bypass route in each workload route table).

✓ Then enable the connector

As with Terraform — enable the GWLB connector on each engine in the console after the stack completes.

How traffic flows, and AZ affinity

1. **Workload → AZ-local GWLBe.** A workload sends internet-bound traffic; its route table sends `0.0.0.0/0` to the AZ-local GWLB endpoint.
2. **GENEVE → same-AZ engine.** The GWLBe GENEVE-encapsulates the flow (UDP/6081) to the GWLB, which forwards it to the engine in the same AZ (cross-zone off → AZ-sticky).
3. **Decap → policy → SNAT.** The sidecar decapsulates onto `efz-gwi0`; the engine evaluates the inner packet through its policy, then SNATs it to the engine's Elastic IP and egresses via the appliance IGW.
4. **Return, re-encapsulated.** Return traffic hits the engine's EIP, is un-NAT'd, and a flow-tag route diverts it to `efz-gwo0`, where the sidecar re-encapsulates the original GENEVE back via GWLB → GWLBe → the originating workload.

◆ **AZ affinity (cross-zone off)**

`cross_zone_lb = false` keeps each flow in its AZ: AZ-b's workloads egress via the AZ-b engine's EIP, AZ-c's via AZ-c, and so on. Benefits: no cross-AZ data-transfer cost, lower latency, and an AZ-isolated blast radius — losing one engine only affects its own AZ; GWLB won't fail those flows over.

Hardening, source/dest check & teardown

◆ Host-firewall hardening

When the GWLB connector is enabled, the engine scopes its inbound GENEVE (UDP/6081) and health-check (TCP/9092) accepts to the VPC CIDR (auto-derived), not `0.0.0.0/0` — so neither port is internet-reachable. Defence-in-depth on top of the security group.

◆ Why source/destination check is disabled

GWLB delivers the real workload packets (decapsulated from GENEVE) to the engine, whose source/destination is another host. EC2's source/destination check would drop those; disabling it (`source_dest_check = false`) is what lets the engine forward and NAT transit traffic. Mandatory for any inline inspection appliance.

TEARDOWN

The engines deregister themselves on terminate. Remove any leftover firewall entries from the console afterwards.

bash — teardown

```
# Terraform
$ cd terraform
$ terraform destroy -var deployment_key=efz_xxxxxxxxxxxxxxx -var ssh_key_name=my-key

# CloudFormation
$ aws cloudformation delete-stack --stack-name enforza-gwlb-deployment-key
```