SCENARIO 01 · SINGLE GATEWAY · DEPLOYMENT-KEY

Deploy a single Enforza gateway on AWS

A minimal, production-shaped landing zone for the Enforza NVA — one VPC, one firewall in a public subnet, and two private workload subnets whose internet-bound traffic is routed through the firewall for secure NAT and L7 egress filtering. Deploy it with Terraform or CloudFormation — including a click-by-click console upload if you'd rather not touch a command line.

◆ Deployment-key variant

The firewall launches from a stock base AMI (latest Amazon Linux 2023) and bootstraps the Enforza engine from the public CDN at first boot using a one-time deployment key. No AWS Marketplace subscription and no Marketplace AMI to manage. Terraform and CloudFormation build the identical architecture.

YOU'LL NEED

- An Enforza deployment (registration) key — one per firewall (see page 4).
- An AWS account, a region, and credentials configured.
- `Terraform ≥ 1.5` (AWS provider ~> 5.0) or the AWS CLI / Console for CloudFormation.

The architecture

One VPC (`10.0.0.0/16`) with an Internet Gateway. A public subnet hosts the Enforza firewall with an Elastic IP; two private subnets (one per AZ) send `0.0.0.0/0` to the firewall's network interface, so workload egress is filtered and SNAT'd on the way out.

What it deploys, and how it boots

THE RESOURCES

- One VPC (`10.0.0.0/16`) and one Internet Gateway.
- One public subnet (`10.0.0.0/24`, AZ 0) hosting the Enforza firewall with an Elastic IP; public IPs auto-assign at launch so the box has outbound internet for its first-boot bootstrap.
- Two private workload subnets (`10.0.1.0/24` AZ 0, `10.0.2.0/24` AZ 1).
- One EC2 instance (default `c6i.large`) on a stock base AMI, source/destination check disabled, that bootstraps the engine at boot.
- A public route table (`0.0.0.0/0 → IGW`) and two private route tables (`0.0.0.0/0 → the firewall's network interface`).

All resources are tagged `Project=enforza` and `Scenario=01-single-gateway-deployment-key`.

How the first-boot bootstrap works

- The instance boots from a stock base AMI — no Enforza software baked in.
- cloud-init runs the user-data script, which fetches the public installer and passes it your deployment key.
- The installer sets up prerequisites and the Enforza engine, which registers with the Enforza cloud using the key.
- The key is consumed at this point (one-time-use), and the firewall appears in the console, ready to claim.

EXPLANATION — The installer targets prod (`https://dl.neon.efz.io/install.sh`). The networking is identical to the Marketplace variant; only the image and first-boot bootstrap differ.

PREREQUISITE

Get a deployment key

1 Generate a single-use key in the console

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

- Generate a key — choose Single-use (one firewall per key)
- Copy it; it looks like `efz_xxxxxxxxxxxxxxx`
- You'll pass it to the deploy command (or paste it as a stack parameter)

EXPLANATION — This is the same deployment key used in the Enforza AWS Lab guide (Part 3). It's a one-time token that lets this firewall prove it may join your account; the engine swaps it for a long-lived licence at first boot, so the key is spent once used. Generate a fresh key per instance — a single-use key is consumed by the first engine that registers with it.

△ One key, one firewall

Deploying two gateways? Mint two keys. Re-running a deploy against an already-registered firewall doesn't need a new key; a brand-new instance does.

OPTION A · TERRAFORM

Deploy with Terraform

2 Init, plan, apply

From the repo, work in the `terraform/` directory and pass your deployment key as a variable.

bash — terraform

```
$ cd terraform
$ terraform init
$ terraform plan  -var deployment_key=efz_xxxxxxxxxxxxxxx
$ terraform apply -var deployment_key=efz_xxxxxxxxxxxxxxx
```

EXPLANATION — `init` downloads the AWS provider, `plan` shows exactly what will be created, and `apply` builds it. The whole VPC, subnets, route tables, firewall instance and Elastic IP come up together; the firewall bootstraps itself on first boot with the key you passed.

3 Optional overrides

Append any of these `-var` flags to `plan` / `apply`:

bash — overrides

```
-var base_ami_id=ami-xxxxxxxx # pin a base image
-var instance_type=c6i.xlarge
-var ssh_key_name=my-key
# CIDRs: vpc_cidr, public_subnet_cidr, private_subnet_a_cidr, private_subnet_b_cidr
```

OPTION B · CLOUDFORMATION (CLI)

Deploy with the AWS CLI

2 Deploy the stack

One self-contained template builds the identical architecture. Pass your key as a parameter override.

bash — aws-cli

```
$ aws cloudformation deploy \
--template-file cloudformation/template.yaml \
--stack-name enforza-single-gateway-deployment-key \
--parameter-overrides DeploymentKey=efz_xxxxxxxxxxxxxxx \
--capabilities CAPABILITY_NAMED_IAM
```

EXPLANATION — `CAPABILITY_NAMED_IAM` is required because the stack creates a named IAM role for the instance. The command uploads the template, creates the stack, and waits — when it returns, the gateway is building. Optional parameters: `BaseAmiId`, `InstallUrl`, `InstanceType`, `SshKeyName`, and the CIDR parameters (`VpcCidr`, `PublicSubnetCidr`, `PrivateSubnetACidr`, `PrivateSubnetBCidr`).

◆ **Prefer clicking to typing?**

The next page walks the same template through the AWS Console — download the file, upload it to CloudFormation, and fill the parameters in a form. No command line needed.

OPTION B · CLOUDFORMATION (CONSOLE — SEMI-CLICKOPS)

Upload the template in the AWS Console

1. **Download the template.** Grab `cloudformation/template.yaml` from the scenario repo and save it locally.
2. **Open CloudFormation.** AWS Console → CloudFormation → Create stack → `With new resources (standard)`.
3. **Choose the template.** Prepare template → `Template is ready`. Specify template → `Upload a template file` → Choose file → select `template.yaml` → Next.
4. **Name it and set parameters.** Stack name `enforza-single-gateway-deployment-key`. Under Parameters, paste `DeploymentKey = efz_xxxxxxxxxxxxxxx`. Leave the rest at defaults (or set `InstanceType`, `SshKeyName`, `InstallUrl`, CIDRs) → Next.
5. **Stack options.** Nothing required — tags are already baked in by the template → Next.
6. **Acknowledge IAM & submit.** On Review, tick "I acknowledge that AWS CloudFormation might create IAM resources with custom names" → Submit.
7. **Watch it build.** The Events tab streams progress to CREATE_COMPLETE. The Outputs tab then shows `FirewallInstanceId` — the instance you'll claim in the console.

EXPLANATION — The acknowledgement is the console's equivalent of the CLI's `--capabilities CAPABILITY_NAMED_IAM` — CloudFormation won't create a named IAM role without your explicit consent. Everything else is the same stack the CLI would deploy, just driven through a form.

AFTER THE DEPLOY

Claim the firewall, and how traffic flows

✓ Claim the firewall

Once the engine has registered (a minute or two after boot), it appears in the Enforza console under `Firewalls`, where you can claim and manage it. The `firewall_instance_id` / `FirewallInstanceId` output identifies the instance.

→ How traffic flows end-to-end

- A workload in a private subnet sends traffic to the internet.
- Its private route table sends `0.0.0.0/0` to the firewall's network interface — not to an IGW or NAT gateway.
- The Enforza engine inspects and filters the egress at Layer 7, then SNATs the packet to the firewall's Elastic IP.
- The public route table sends `0.0.0.0/0` to the Internet Gateway, and the packet leaves.
- Return traffic follows the reverse path back through the firewall to the workload.

Source/dest check & teardown

◆ Why source/destination check is disabled

By default EC2 enforces a source/destination check: an instance only accepts packets whose source or destination is its own address, and silently drops everything else. A NAT/forwarding appliance exists precisely to handle packets addressed to other hosts (the private workloads). Disabling the check (`source_dest_check = false` in Terraform, `SourceDestCheck: false` in CloudFormation) is what permits the instance to forward those packets. Without it, the gateway would drop all transit traffic and nothing would reach the internet.

TEARDOWN — STOP THE METER

Remove everything when you're done. Then delete the firewall from the Enforza console.

bash — teardown

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

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

✓ Done

Same architecture whichever path you took — one gateway doing secure NAT and L7 egress filtering for two private workload subnets. To scale out, add private subnets and point their route tables at the same firewall ENI, or deploy another gateway with a fresh key.