HANDS-ON LAB · FROM AN EMPTY ACCOUNT

Build an Enforza AWS lab from scratch

Click-by-click, from nothing in AWS to a working Enforza firewall — enrolled in the Cloud Controller, enforcing a policy, and showing you live traffic. Every step is written for a semi-technical reader, with a plain-English explanation of what you're doing and why. About 20 minutes; a `t3.small` costs a few pence an hour.

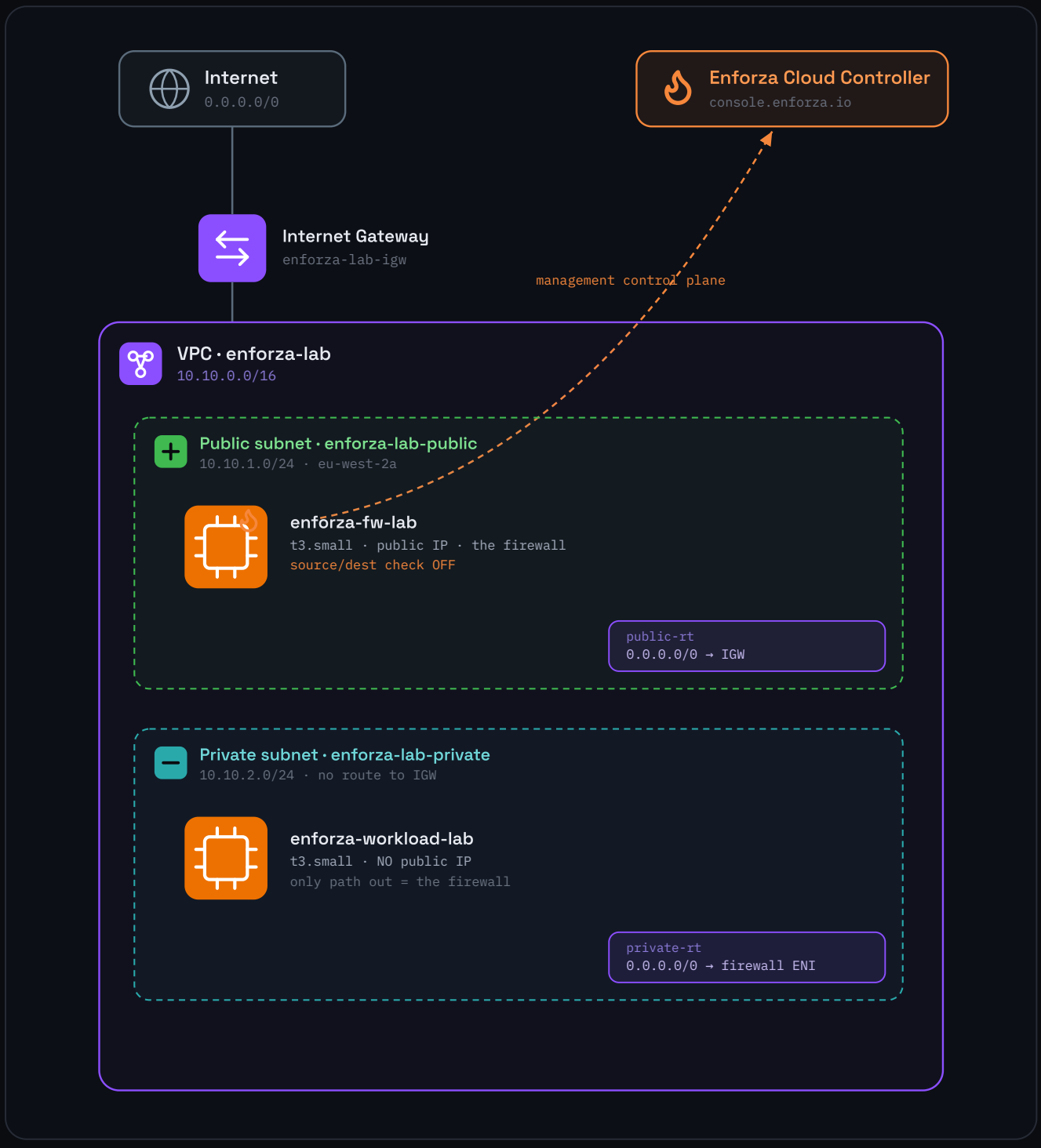
> **△ This is a lab build, kept deliberately simple**
>
> One VPC, a public subnet holding the EC2 that is the firewall, a private subnet holding a workload whose traffic is routed through it, and a wide-open security group. Perfect for learning end-to-end — not a production topology, but the private-subnet-through-firewall shape is exactly the real inspection pattern.

BEFORE YOU START

- An AWS account with permission to create VPC/EC2 resources.
- An Enforza console (CCX) login with at least 1 free engine licence.
- An SSH key pair (create or reuse at launch). Region: this guide uses `eu-west-2` — any works, just stay consistent.

What you'll build

An Internet Gateway connects the VPC to the internet. The public subnet holds the firewall EC2 (public IP, source/dest check off), which enrols with the Cloud Controller. The private subnet holds a workload with no public IP whose route table sends `0.0.0.0/0` to the firewall's ENI — forcing its traffic through the firewall.



Networking — VPC · IGW · route table Compute — EC2 Public subnet Private subnet

PART 1 · BUILD THE NETWORK FOUNDATION

Create the VPC and its public subnet

1 Create a VPC

Console: VPC → Your VPCs → Create VPC → choose VPC only →

- Name: `enforza-lab`
- IPv4 CIDR: `10.10.0.0/16`
- Leave IPv6 off, tenancy default → Create VPC

EXPLANATION — A VPC is your own private, isolated network inside AWS — think of it as the "building" all your lab machines live in. The CIDR `10.10.0.0/16` is the pool of private IP addresses available inside it (65k addresses). Nothing here is reachable from the internet yet; we add that next.

2 Create a public subnet

Console: VPC → Subnets → Create subnet →

- VPC: `enforza-lab`
- Name: `enforza-lab-public`
- Availability Zone: pick one, e.g. `eu-west-2a`
- IPv4 CIDR: `10.10.1.0/24` → Create subnet

EXPLANATION — A subnet is a slice of the VPC's addresses pinned to one Availability Zone (one physical data-centre area). We'll put the firewall here. It's called "public" only because in Step 4 we give it a route to the internet — a subnet isn't public until its route table says so.

PART 1 · BUILD THE NETWORK FOUNDATION

Add the gateway and route to the internet

3 Create & attach an Internet Gateway

Console: VPC → Internet gateways → Create internet gateway →

- Name: `enforza-lab-igw` → Create
- Then Actions → Attach to VPC → select `enforza-lab` → Attach

EXPLANATION — An Internet Gateway (IGW) is the door between your VPC and the public internet. Creating it isn't enough — you must attach it to the VPC, and then (next step) tell the subnet's route table to use it. The firewall needs internet access so it can reach the Enforza control plane to enrol.

4 Route the subnet to the internet

Console: VPC → Route tables → Create route table →

- Name: `enforza-lab-public-rt`, VPC: `enforza-lab` → Create
- Routes tab → Edit routes → Add route: `0.0.0.0/0` → Target Internet Gateway → `enforza-lab-igw` → Save
- Subnet associations tab → Edit → tick `enforza-lab-public` → Save

EXPLANATION — A route table is the signpost that tells traffic where to go. The `0.0.0.0/0` route means "anything not local to the VPC → send to the internet gateway". Associating it with the public subnet is what actually makes that subnet "public". Without this, your firewall would launch but never be able to phone home to Enforza.

PART 1 · BUILD THE NETWORK FOUNDATION

Open a security group (lab only)

5 Create a wide-open security group

Console: VPC (or EC2) → Security groups → Create security group →

- Name: `enforza-lab-any`, VPC: `enforza-lab`
- Inbound rules → Add rule: Type `All traffic`, Source `Anywhere-IPv4 (0.0.0.0/0)`
- Outbound rules: leave the default `All traffic → 0.0.0.0/0`
- Create security group

EXPLANATION — A security group (AWS's equivalent of an NSG) is a stateful firewall around the instance's network card. We open it any/any so it never gets in the way while you're learning — that way, the only thing filtering traffic is Enforza itself, which is the whole point of the lab. Never do this in production — there you'd lock inbound down to just your admin IP. Note that even wide-open, AWS security groups only allow; the Enforza engine is what gives you deny and hostname rules.

PART 2 · LAUNCH THE FIREWALL EC2

Launch the instance that is the firewall

6 Launch the instance

Console: EC2 → Instances → Launch instances →

- Name: `enforza-fw-lab`
- AMI: `Ubuntu Server 22.04 LTS` (Debian/RHEL/Rocky/Amazon Linux 2023 also work)
- Instance type: `t3.small` (2 vCPU / 2 GiB — fine for a lab)
- Key pair: select an existing one, or Create new key pair (download the `.pem`)
- Network settings → Edit: VPC `enforza-lab`, subnet `enforza-lab-public`, Auto-assign public IP: Enable, security group `enforza-lab-any`
- Launch instance. Note the Instance ID and, once running, its Public IPv4 address

EXPLANATION — This EC2 instance is the Enforza firewall — the engine is a single Linux daemon we'll install on it in Part 3. Auto-assign public IP is essential: it's how the box reaches the internet (via the IGW + route table you just built) to enrol and stream logs. The key pair is your SSH login. We picked a small size because a lab pushes tiny traffic; the engine auto-tunes to bigger instances when you need throughput.

PART 2 · LAUNCH THE FIREWALL EC2

Turn off the source/destination check

7 Disable source/dest check ⚠ critical

Console: EC2 → Instances → select `enforza-fw-lab` →

- Actions → Networking → Change source/destination check
- Tick Stop (uncheck the enabled box) → Save

```
# CLI equivalent, region eu-west-2
aws ec2 modify-network-interface-attribute --region eu-west-2 \
  --network-interface-id <eni-of-the-instance> --no-source-dest-check
```

EXPLANATION — By default AWS drops any packet whose destination isn't the instance's own IP address. But a firewall's entire job is to handle packets meant for other machines. So this AWS safety check must be off, or the firewall will silently blackhole every forwarded packet. This is the single most common "it's installed but nothing works" mistake. In a lab where the box only filters its own traffic you can get away without it, but turn it off now so the optional Part 7 (routing a workload through it) just works.

PART 3 · ENROL THE ENGINE WITH THE CONSOLE

Get a deployment key

8 Get a deployment key

Console (CCX): sidebar → Onboard Firewall → Deployment Keys tab → Generate key. Choose:

- Single-use — binds exactly one firewall (use this for the lab), or
- Provisioning — reusable across a fleet (Terraform/Ansible/CI)
- Copy the install command it shows you

```
curl -fsSL https://dl.neon.efz.io/install.sh \
  | sudo bash -s -- --regkey=YOUR-KEY-HERE
```

EXPLANATION — The deployment key is a one-time token that lets this firewall prove to the Enforza cloud "I'm allowed to join your account." The install command downloads the engine installer and passes the key to it. A single-use key is consumed by the first engine that uses it — so mint one key per firewall. (There's also a Cloud Claim tab for AWS Marketplace instances that self-register — ignore it for a manual lab install.)

PART 3 · ENROL THE ENGINE WITH THE CONSOLE

Install, then verify it's online

9 SSH in and run the install command

From your laptop: use the `.pem` from Step 6 and the instance's public IP, then paste the copied install command.

```
# first time only
chmod 400 enforza-lab.pem
ssh -i enforza-lab.pem ubuntu@<PUBLIC-IP>
# then paste the copied install command
curl -fsSL https://dl.neon.efz.io/install.sh | sudo bash -s -- --regkey=YOUR-KEY-HERE
```

EXPLANATION — SSH is a secure remote terminal into the box. The installer (~30–60 s) detects your Linux distro, installs prerequisites, fetches the engine binary, swaps your one-time key for a long-lived licence token, and starts the `enforza-engine` service. The username is `ubuntu` for the Ubuntu AMI (`admin` for Debian, `ec2-user` for Amazon Linux/RHEL).

10 Verify it's online

On the VM: `systemctl status enforza-engine` → should say active (running). **In the console:** sidebar → Firewalls → your engine appears Online within ~10 seconds.

EXPLANATION — "Online" means the engine has enrolled and opened its always-on WebSocket back to the Enforza cloud. That control channel is how the cloud pushes policy and how the engine streams logs — and it bypasses policy by design, so you can never accidentally lock the engine away from its own management plane. If it never shows up, the box has no internet path — re-check the IGW, route table, and public IP (Part 1 + Step 6).

PART 4 · BUILD A BASIC POLICY

Three sections = input / forward / output

ENFORZA SECTION	CLASSIC	WHAT IT CONTROLS
to-firewall	INPUT	Traffic addressed **to the firewall box itself** (e.g. SSH to it)
through-firewall	FORWARD	Traffic **passing through** the firewall between other hosts — the main event
from-firewall	OUTPUT	Traffic the firewall box **originates itself** (DNS, NTP, updates)

EXPLANATION — Every policy is built in the console — no YAML editing. Each section has its own **default action**; we'll set them to **drop** (fail-closed: if nothing explicitly allows a packet, it's denied). Nothing reaches the firewall until you **Push** — saving a draft is safe.

11 Create the policy shell

Console: sidebar → Policies → New policy →

- Name: `lab-policy`
- Default action: drop → Create

EXPLANATION — This creates an empty policy with a deny-by-default posture. From here we add just enough "allow" rules to keep things working, then one "deny" rule to prove filtering works.

PART 4 · BUILD A BASIC POLICY

Let the OS out, keep your SSH

12 from-firewall (OUTPUT) — let the engine's OS reach out

Open the Local tab (`from-firewall`) → Add rule for each:

COMMENT	DESTINATION	PROTOCOL	PORT	ACTION
dns	any	udp/tcp	53	Accept
ntp	any	udp	123	Accept
https	any	tcp	443	Accept

EXPLANATION — These let the firewall's own operating system do housekeeping — resolve names (DNS), keep its clock right (NTP), and fetch updates (HTTPS). Everything the engine needs to reach the Enforza cloud already bypasses policy, so the box won't brick without these — but OS services start failing, so it's good hygiene to allow them.

13 to-firewall (INPUT) — keep your SSH, drop the rest

Open the Management tab (`to-firewall`) → Add rule `allow my ssh`: Source `<your-laptop-public-IP>/32`, Protocol `tcp`, Port 22, Accept. Leave the section default action = drop.

EXPLANATION — This section guards the firewall box itself. We explicitly allow SSH only from your IP so you keep your terminal, and drop everything else aimed at the box. ⚠ If you skip the SSH allow and set default drop, your next SSH attempt is blocked (the engine's own cloud control channel keeps working — only your SSH is affected). Find your public IP at `ifconfig.me`. Note: hostname/URL rules are not allowed in this section — inbound-to-the-box traffic is matched on IP/port only.

PART 4 · BUILD A BASIC POLICY

through-firewall — the actual inspection

14 Add two forwarding rules

Open the Network tab (`through-firewall`) → add two rules, then Save.

COMMENT	SOURCE	DEST	PROTO	PORT	ACTION	FLAGS
web egress	any	any	tcp	443	Accept	—
block fwd ssh	any	any	tcp	22	Drop	tick Log

EXPLANATION — This is the section that inspects traffic passing through the firewall — the real product. We allow HTTPS so normal web traffic flows, and we drop + log SSH so you have something visibly blocked to watch in the live logs later. Rules are first-match-wins within a section — drag the row handle to reorder. Ticking Log is what makes a matched flow show up in Live Traffic.

Want hostname filtering like `**.github.com` instead of IP/port? Switch the rule type to URL Filtering — see the companion egress-filtering guide.

PART 4 → 5 · PUBLISH, THEN BIND

Push the policy, then wire it to the firewall

15 Push the policy live

Console: from the policy editor toolbar → Push. The publish dialog runs preflight checks (schema + any guardrails), then publishes.

EXPLANATION — Push is the moment your draft becomes a real, versioned policy in the cloud — but it's still not enforcing on any firewall yet. Publishing and binding (next) are two separate steps on purpose: you can prepare a policy safely without touching a live box. Tip: Download policy first if you want to eyeball the generated YAML.

16 Apply the policy (bind it)

Console: sidebar → Bindings → find your `enforza-fw-lab` row (shows Unbound) → Apply policy → pick `lab-policy` → Confirm.

EXPLANATION — Publishing made the policy available; the binding is what actually wires this firewall to this policy. A brand-new engine is Unbound — it enforces nothing and drops nothing until you bind. The engine notices the change on its next config poll (within ~5 seconds) and applies it.

PART 5 · BIND THE POLICY TO YOUR FIREWALL

Confirm it applied

17 Confirm it applied

Console: sidebar → Firewalls. Your firewall row now shows the policy name and a green Synced chip with the active policy's SHA.

EXPLANATION — The green Synced chip means the engine successfully downloaded, compiled, and loaded your policy — it is now live and enforcing. If instead you see a red Parse / Compile / Apply error chip, it carries the engine's own error message — fix the rule and Push again.

✓ Checkpoint

Engine Online, policy Synced. The firewall is now enforcing `lab-policy` on its own traffic. Next you'll watch it work — then force a real workload through it.

PART 6 · VIEW THE TRAFFIC

Watch policy decide, live

18 Watch live logs

Console: Firewalls → click your firewall → Live logs tab (or sidebar → Live Traffic to stream several together). Generate traffic to see it — SSH into the box and run `curl -sI https://example.com` → should succeed (accept on TCP/443).

EXPLANATION — Every accept / drop / flow-close on the engine streams to the console over the WebSocket only while a viewer is open (to keep costs bounded; each session caps at 15 minutes — just reload to continue). New events appear at the top. Pause to inspect, Resume to catch up, Clear to wipe the on-screen buffer.

19 Read a row

Columns: `Time` · `Flow` · `Action` · `Proto` · `Source` · `Destination` · `SNAT` · `Hostname` · `Rule`. Click any row for the full JSON detail (Event / Network / GeoIP / Packet / TLS / Flow / Engine).

EXPLANATION — This closes the whole loop: you can literally watch your `web egress` rule turn a request green and — if you try to SSH through the box — your `block fwd ssh` rule turn it red. The Rule column tells you exactly which line of policy decided each packet, which is how you debug a policy in seconds instead of guessing.

PART 6 → 7 · HISTORY, THEN ROUTE A REAL WORKLOAD

Search the past, add a private subnet

20 Search historical traffic

For "what happened earlier", use the firewall's `Search logs` tab: pick a time range (up to 7 days) and filter by action, protocol, IP, port, `Hostname / SNI`, or free text.

EXPLANATION — Live logs only show new events. Search logs queries the engine's own on-disk history — nothing is stored in the cloud by default, so your traffic data stays on your VM unless you set up Log Export (S3 / Splunk / Sentinel) under Configure → Log Export.

21 Create the private subnet

Console: VPC → Subnets → Create subnet →

- VPC: `enforza-lab`, Name: `enforza-lab-private`
- Same AZ as the firewall (e.g. `eu-west-2a`)
- IPv4 CIDR: `10.10.2.0/24` → Create subnet

EXPLANATION — This subnet has no route to the Internet Gateway — that's deliberate. A machine here can't reach the internet on its own; we'll force its traffic through the firewall instead. That's what makes the inspection real: the workload literally cannot bypass the firewall.

PART 7 · ROUTE A REAL WORKLOAD THROUGH THE FIREWALL

Point the private subnet at the firewall's ENI

22 Find the firewall's network interface (ENI)

Console: EC2 → Instances → `enforza-fw-lab` → Networking tab → Network interfaces → note the Interface ID (`eni-…`).

> **EXPLANATION —** The ENI is the firewall's virtual network card. We'll point the workload's default route at this specific card so all its traffic lands on the firewall. (Full detail on locating the right ENI in multi-NIC setups is in the companion route-a-subnet guide.)

23 Route the private subnet through the firewall ENI

Console: VPC → Route tables → Create route table →

- Name: `enforza-lab-private-rt`, VPC: `enforza-lab` → Create
- Routes tab → Edit routes → Add route: `0.0.0.0/0` → Target Network Interface → the firewall's `eni-…` → Save
- Subnet associations tab → Edit → tick `enforza-lab-private` → Save

> **EXPLANATION —** This is the heart of the inspection pattern. The private subnet's default route now says "anything bound for the internet → hand it to the firewall's ENI." The firewall inspects it against your `through-firewall` rules, then source-NATs (SNAT) it out through its own public path. Remember Step 7: if the firewall's source/dest check were still on, AWS would drop these forwarded packets — that's why we turned it off up front.

PART 7 · ROUTE A REAL WORKLOAD THROUGH THE FIREWALL

Launch the workload behind the firewall

24 Launch the workload EC2 in the private subnet

Console: EC2 → Launch instances →

- Name: `enforza-workload-lab`, same Ubuntu AMI, `t3.small`
- Network settings → Edit: VPC `enforza-lab`, subnet `enforza-lab-private`
- Auto-assign public IP: Disable, security group `enforza-lab-any`
- Same key pair → Launch

EXPLANATION — This is the "customer" machine sitting behind the firewall. It has no public IP, so its only path to the internet is the route you just made — through the firewall. To get a terminal on it (it's not directly reachable), either use it via AWS Systems Manager → Session Manager, or SSH to it from the firewall box as a jump host (`ssh ubuntu@10.10.2.x` from inside the firewall). Session Manager works because the workload's outbound 443 now flows out through the firewall, which your `web egress` rule allows.

PART 7 · ROUTE A REAL WORKLOAD THROUGH THE FIREWALL

Generate traffic and watch it on the firewall

25 Generate traffic and watch it

- In the console, open Firewalls → `enforza-fw-lab` → Live logs (leave it running).
- On the workload box, generate some traffic:

```
curl -sI https://example.com     # allowed  → accept on TCP/443
nc -vz somehost 22               # crosses the firewall → block-fwd-ssh Drop fires
```

Watch the rows appear on the firewall's live log — the workload's private IP (`10.10.2.x`) as Source, the Rule column naming which policy line decided each flow, and the SNAT column green as the firewall translates the traffic on the way out.

EXPLANATION — This closes the loop end-to-end: a real machine's traffic is now being forced through your firewall, inspected against the `through-firewall` rules, and shown to you live with the deciding rule named. Change a rule, Push, re-run the `curl`, and watch the verdict flip — that's the full authoring-to-observation cycle in seconds.

This is the real inspection pattern: the workload has no direct internet path, so all its outbound traffic is forced through the firewall. Now the live logs fill with the workload's real web traffic — and your FQDN allow-lists actually protect something.

Troubleshooting & clean-up

SYMPTOM	MOST LIKELY CAUSE
Install exits `registration key not found`	Key already consumed or expired — mint a fresh one (Onboard Firewall → Deployment Keys).
Engine installs but never appears in console	No outbound internet — check IGW attached, route `0.0.0.0/0 → IGW`, subnet association, and Auto-assign public IP.
Firewall shows red Apply/Compile chip	Policy rejected by the engine — read the inline error, fix the rule, Push again.
Traffic through the box is dropped unexpectedly	Source/dest check still on (Step 7), or a section default-drop with no matching allow.
You got locked out of SSH	`to-firewall` default-drop with no `allow my ssh` rule — fix via Session Manager or relax the rule and re-push.
Nothing in Live logs	No viewer open when traffic flowed, the flow didn't match a Log-ticked rule, or no traffic was generated.

△ Clean up — stop the meter

When you're done: EC2 → Instances → select `enforza-fw-lab` → Terminate. Then delete the workload instance, the security group, subnets, route tables, IGW, and the VPC. In the console the firewall drops to Offline — remove it from Firewalls if you won't reuse it. A running `t3.small` costs a few pence per hour; terminating stops all charges. Your deployment key was single-use and is already spent — mint a new one next time.

One-glance recap

1. **Network.** VPC → public subnet → IGW → route `0.0.0.0/0 → IGW` → any/any SG.
2. **Instance.** Launch in the public subnet, public IP on, source/dest check OFF.
3. **Enrol.** Onboard Firewall → Deployment Key → SSH → paste install command → Online.
4. **Policy.** `drop` default; allow OS egress (from-firewall), guard SSH (to-firewall), inspect + block (through-firewall) → Push.
5. **Bind.** Bindings → Apply policy → Synced.
6. **Observe.** Firewalls → Live logs (and Search logs for history).
7. **Route a workload.** Private subnet (no IGW) → route `0.0.0.0/0 → firewall ENI` → watch its real traffic inspected live.

✓ You've done the full loop

Empty account → a firewall enrolled, enforcing a deny-by-default policy, and showing live traffic with the deciding rule named — plus a real workload forced through it. From here, swap the `web egress` rule for an FQDN allow-list and you're running the production pattern.

◆ Where next

Egress filtering guide — L3/L4 & FQDN/SNI rules and the FinOps play. Route-a-subnet guide — the routed-NVA pattern in production detail.