ITI8801 · Practice 3 · AWS

Week 2's setup as code, built in seven steps

Siim Rebane · TalTech · Autumn 2026

The task

Build, on your own AWS account, with Terraform and nothing else:

- the **network** of week 2: a VPC, an app subnet with two app instances running the course beacon, an application load balancer as the only door
- a **managed PostgreSQL** instance (RDS) in two private subnets, reachable only from the app subnet
- a **NAT gateway** as the way out during the build, removed by one variable
- a **bastion** in a public subnet as the only SSH path; no app instance ever has a public IP

Then: the assignment 3 validator green, `terraform destroy`, an empty account.

Rule of the day: **nobody applies a plan they have not read.**

How today works: one file at a time

Seven files in a `steps/` folder. Terraform reads every `.tf` file in the working directory and nothing in subfolders, so a step is: copy one file up, `plan`, read, `apply`, look at what appeared.

```
cp steps/0N-something.tf . && terraform plan -var-file=me.tfvars && terraform apply -var-file=me.tfvars
```

Step	File	Adds	Plan
1	`01-network.tf`	VPC, internet gateway, five subnets, public route table	10 to add
2	`02-walls.tf`	four security groups	4
3	`03-load-balancer.tf`	load balancer, target group, listener	3
4	`04-way-out.tf`	elastic IP, NAT gateway, private route table, its default route, associations	7
5	`05-database.tf`	DB subnet group, RDS instance	2
6	`06-app-vms.tf`	key pair, two instances, two target attachments	5
7	`07-bastion.tf`	the bastion instance	1

Final state, and how it is proven

The assignment 3 validator sends requests to the load balancer's DNS name and checks six things live:

1. the address belongs to AWS
2. exactly **two** course beacons answer behind it
3. no beacon can reach the internet
4. port 22 on that address does not answer
5. every beacon has an address in `10.61.0.0/16`
6. every beacon reaches the database, and the database's address is in **`10.61.20.0/23`**

Six green, and a seventh box opens: you upload the configuration files, the ones that built this. Seven green: the check is stored with a code, that code goes to Moodle, and you destroy.

The plan on one slide

Phase	What	Time
0	tools, `aws login`, `terraform init`, `me.tfvars`	10 min
1–4	network, walls, load balancer, way out: four short applies	25 min
5	the database: one apply, six minutes, spent reading step 6	10 min
6	the app instances; the beacon answers through the load balancer	15 min
7	the bastion; SSH inside, the database rule seen from both sides	15 min
8	`locked=true`: plan, read, apply; verify	10 min
9	validator green; `terraform destroy`; empty account	15 min

Before you start: addresses and money

- **Elastic IPs**: AWS allows five per region by default. This configuration uses two, the NAT gateway's and the bastion's. Anything left from week 2 counts too: *EC2 → Elastic IPs*, release what is not in use
- **The NAT gateway is billed by the hour**, about 5 cents, and the Free Plan does not exempt it: it comes out of your credit. It exists from step 4 until the lock-down in step 8, about an hour of today, so a few cents. Left running for a month it is 35 euros
- **RDS** `db.t4g.micro` is inside the Free Plan's allowance. Storage is 20 GB; do not raise it
- **Region**: the one you used in weeks 1 and 2, in `me.tfvars`. Everything today lives in that one region

Nothing from week 1 or 2 needs to survive: A1 was graded from the running instance, A2 from the stored check.

Phase 0 — tools

```
terraform version        # 1.6 or newer; OpenTofu works the same, the command is tofu
aws --version            # 2.32 or newer; an older CLI does not know the next command
aws login                # opens a browser; sign in as you do for the console
aws sts get-caller-identity      # your account id: this is the account that pays
```

`aws login` is the AWS counterpart of `az login`: temporary credentials on the laptop, refreshed while the session lasts, nothing long-lived written anywhere. Get the configuration from **github.com/siimrebane/iti-8801**, then:

```
git clone https://github.com/siimrebane/iti-8801.git
cd iti-8801/terraform/aws
terraform init
```

`init` downloads the aws provider, about 400 MB. Start it now; it runs while we read.

Open `providers.tf` while it runs: the provider block takes `region = var.region` and nothing else. Credentials come from `aws login`, never from the files.

Phase 0 — your variables

Create `me.tfvars` in `terraform/aws`. It is listed in `.gitignore`; check that before you write a password into it:

```
region        = "eu-central-1"              # the region you used in weeks 1 and 2
prefix        = "abc123"                    # your student code, lowercase letters and digits
beacon_image  = "ghcr.io/siimrebane/iti8801-beacon:a3"
db_password   = "<pick one, 12+ characters>"
admin_ssh_key = "ssh-ed25519 AAAA... you@laptop"   # cat ~/.ssh/id_ed25519.pub
my_ip         = "193.40.12.34"              # curl -4 ifconfig.me
```

- `instance_type` has a default, `t3.micro`; change it only if the plan says the type is not offered in your region
- `locked` stays at its default, `false`: the NAT gateway exists during the build
- The beacon tag is `a3`: that build reports the database's address, which check 6 needs

Step 1 — the network: plan

```
cp steps/01-network.tf .
terraform plan -var-file=me.tfvars
```

```
data.aws_availability_zones.azs: Read complete after 0s [id=eu-central-1]
data.aws_ami.ubuntu: Read complete after 0s [id=ami-03f92a7a8a26c81af]

  # aws_internet_gateway.igw will be created
  # aws_route_table.public will be created
  # aws_route_table_association.public[0] will be created
  # aws_route_table_association.public[1] will be created
  # aws_subnet.app will be created
  # aws_subnet.db[0] will be created
  # aws_subnet.db[1] will be created
  # aws_subnet.public[0] will be created
  # aws_subnet.public[1] will be created
  # aws_vpc.vpc will be created

Plan: 10 to add, 0 to change, 0 to destroy.
```

Two `data` blocks were **read** first: the region's zones and the current Ubuntu image. Ten things to create: a VPC `10.61.0.0/16`, an internet gateway, five subnets, a route table and its two attachments.

Step 1 — the network: apply

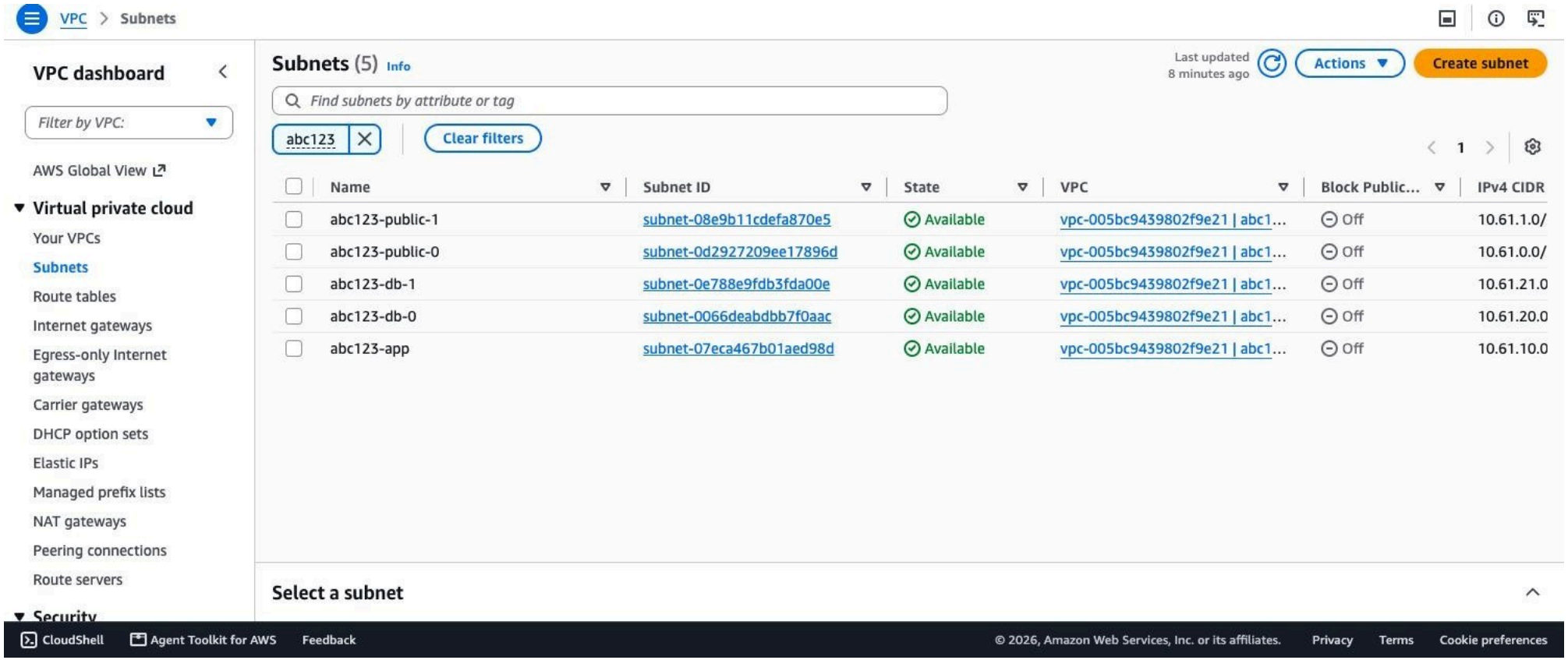
```
terraform apply -var-file=me.tfvars
  Enter a value: yes

aws_vpc.vpc: Creation complete after 2s [id=vpc-005bc9439802f9e21]
aws_internet_gateway.igw: Creation complete after 1s [id=igw-05b4d75839e546b9d]
aws_subnet.app: Creation complete after 1s [id=subnet-07eca467b01aed98d]
aws_subnet.public[1]: Creation complete after 1s [id=subnet-08e9b11cdefa870e5]
aws_route_table.public: Creation complete after 1s [id=rtb-0107dede455997b86]
aws_subnet.public[0]: Creation complete after 4s [id=subnet-0d2927209ee17896d]
aws_subnet.db[1]: Creation complete after 4s [id=subnet-0e788e9fdb3fda00e]
aws_subnet.db[0]: Creation complete after 5s [id=subnet-0066deabdbb7f0aac]
aws_route_table_association.public[0]: Creation complete after 0s [id=rtbassoc-01cdc3c5ce377ee93]
aws_route_table_association.public[1]: Creation complete after 0s [id=rtbassoc-0d96ba82ed0d1c0e2]

Apply complete! Resources: 10 added, 0 changed, 0 destroyed.
```

- Twelve seconds. The VPC first, then everything that needs only the VPC at the same time, then the attachments. The order came from the references in the file
- A new file appeared beside `me.tfvars`: `terraform.tfstate`. Open it once, find a resource's `id`, close it. You will not edit it
- `terraform state list` prints twelve lines: ten resources and the two data sources

Step 1 — in the console



VPC → Subnets, filtered by `abc123`: five subnets in two zones. `public-0` and `public-1`, `app`, `db-0` and `db-1`. *Your VPCs* shows `abc123-vpc` beside the default one.

Step 1 — what you wrote

Subnet	Range	Zone	For
`public[0]`, `public[1]`	`10.61.0.0/24`, `10.61.1.0/24`	a, b	the load balancer needs two zones; the bastion and the NAT gateway sit in the first
`app`	`10.61.10.0/24`	a	the two app instances
`db[0]`, `db[1]`	`10.61.20.0/24`, `10.61.21.0/24`	a, b	RDS insists on subnets in two zones, even for one instance

- `aws_subnet.public`: `count = 2`, `cidr_block = cidrsubnet(aws_vpc.vpc.cidr_block, 8, count.index)`, the n-th /24 of the /16; `availability_zone` from the zones data source, by index
- `map_public_ip_on_launch = false` on the public subnets: nothing gets a public address unless a block asks for one
- `aws_route_table.public`: an inline `route { gateway_id = aws_internet_gateway.igw.id }`, two associations. The app and db subnets use the VPC's main table for now: no way out

Before every step: plan again

```
terraform plan -var-file=me.tfvars
```

```
data.aws_availability_zones.azs: Read complete after 0s [id=eu-central-1]
data.aws_ami.ubuntu: Read complete after 0s [id=ami-03f92a7a8a26c81af]
aws_vpc.vpc: Refreshing state... [id=vpc-005bc9439802f9e21]
aws_internet_gateway.igw: Refreshing state... [id=igw-05b4d75839e546b9d]
aws_subnet.public[0]: Refreshing state... [id=subnet-0d2927209ee17896d]
...
aws_route_table_association.public[1]: Refreshing state... [id=rtbassoc-0d96ba82ed0d1c0e2]

No changes. Your infrastructure matches the configuration.
```

Nothing new in the files, so Terraform read the two data sources again, asked the cloud about its ten ids, compared, and found nothing to do. This is the line to expect **before every step** today: if it ever says something else, something was changed outside the files, and you read the plan before you copy the next file.

Step 2 — the walls: plan

```
cp steps/02-walls.tf .
terraform plan -var-file=me.tfvars
```

```
  # aws_security_group.alb will be created
  # aws_security_group.app will be created
  # aws_security_group.bastion will be created
  # aws_security_group.db will be created

Plan: 4 to add, 0 to change, 0 to destroy.
```

Four security groups. Week 2 had two; the bastion and the database each get their own. A security group denies everything not listed, so there is no deny rule to write anywhere.

Step 2 — the walls: apply

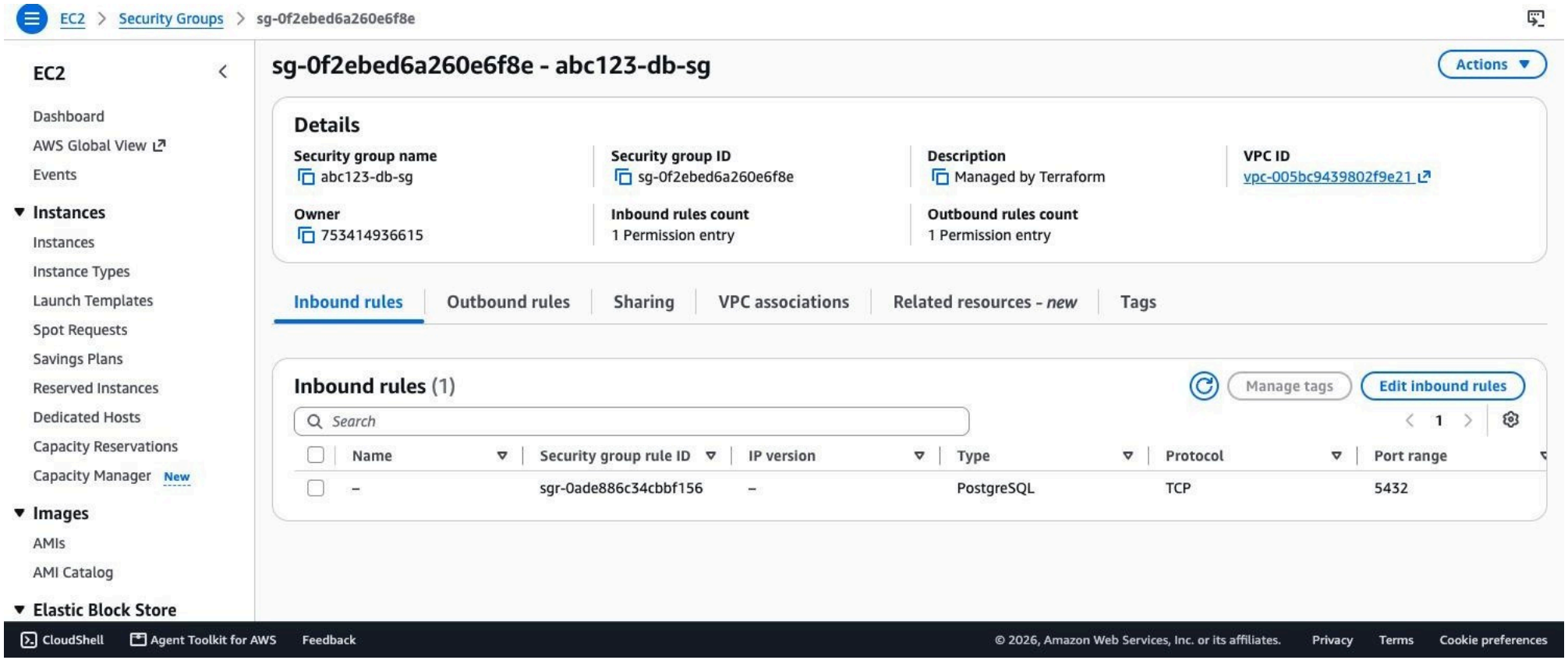
```
terraform apply -var-file=me.tfvars
  Enter a value: yes

aws_security_group.bastion: Creation complete after 3s [id=sg-0a8c1a520fc5d7802]
aws_security_group.alb: Creation complete after 3s [id=sg-08c2751443a16ce74]
aws_security_group.app: Creation complete after 3s [id=sg-027a2ce201d76aafe]
aws_security_group.db: Creation complete after 3s [id=sg-0f2ebed6a260e6f8e]

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.
```

- Fifteen seconds, all four at once: a group that names another group as a source is created after it, and Terraform sorted that out
- `terraform state list | wc -l` says 16. `terraform plan -var-file=me.tfvars`: *No changes*

Step 2 — in the console



EC2 → Security Groups → `abc123-db-sg` *→ Inbound rules*: one row, PostgreSQL 5432, source the app group's id. No rule names the bastion: that is the whole database firewall.

Step 2 — what you wrote

Group	Rule	Why
`alb`	80 from `0.0.0.0/0`	the load balancer is the door
`app`	80 from the `alb` group	only the load balancer delivers on port 80
`app`	22 from the `bastion` group	SSH only from the bastion, by group, not by address
`db`	5432 from the `app` group	PostgreSQL from the app instances only
`bastion`	22 from `<your IP>/32`	one laptop, and no other address

- Each `aws_security_group` holds `ingress { ... }` and `egress { ... }` blocks: the *Inbound rules* table, row by row
- `security_groups = [aws_security_group.alb.id]` on the app group: the source is **membership in another group**, not an address
- `cidr_blocks = ["${var.my_ip}/32"]` on the bastion group: your variable inside a string. New IP tomorrow: one line, `apply`, *1 to change*
- Every group allows everything out; whether a packet gets out is the route table's decision

Step 3 — the door: plan

```
cp steps/03-load-balancer.tf .
terraform plan -var-file=me.tfvars
```

```
  # aws_lb.lb will be created
  # aws_lb_listener.http will be created
  # aws_lb_target_group.app will be created

Plan: 3 to add, 0 to change, 0 to destroy.
```

Week 2's application load balancer in three blocks: the load balancer in the two public subnets, an empty **target group** with a health check on `/hello`, and the **listener** that forwards port 80 to the group.

Step 3 — the door: apply

```
terraform apply -var-file=me.tfvars
  Enter a value: yes

aws_lb_target_group.app: Creation complete after 1s [id=…/targetgroup/abc123-app-tg/9705b03f0156ad84]
aws_lb.lb: Creation complete after 2m12s [id=…/loadbalancer/app/abc123-alb/452f8dcfb9b166a6]
aws_lb_listener.http: Creation complete after 0s [id=…/listener/app/abc123-alb/452f8dcfb9b166a6/e6e698e7e16a9e68]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

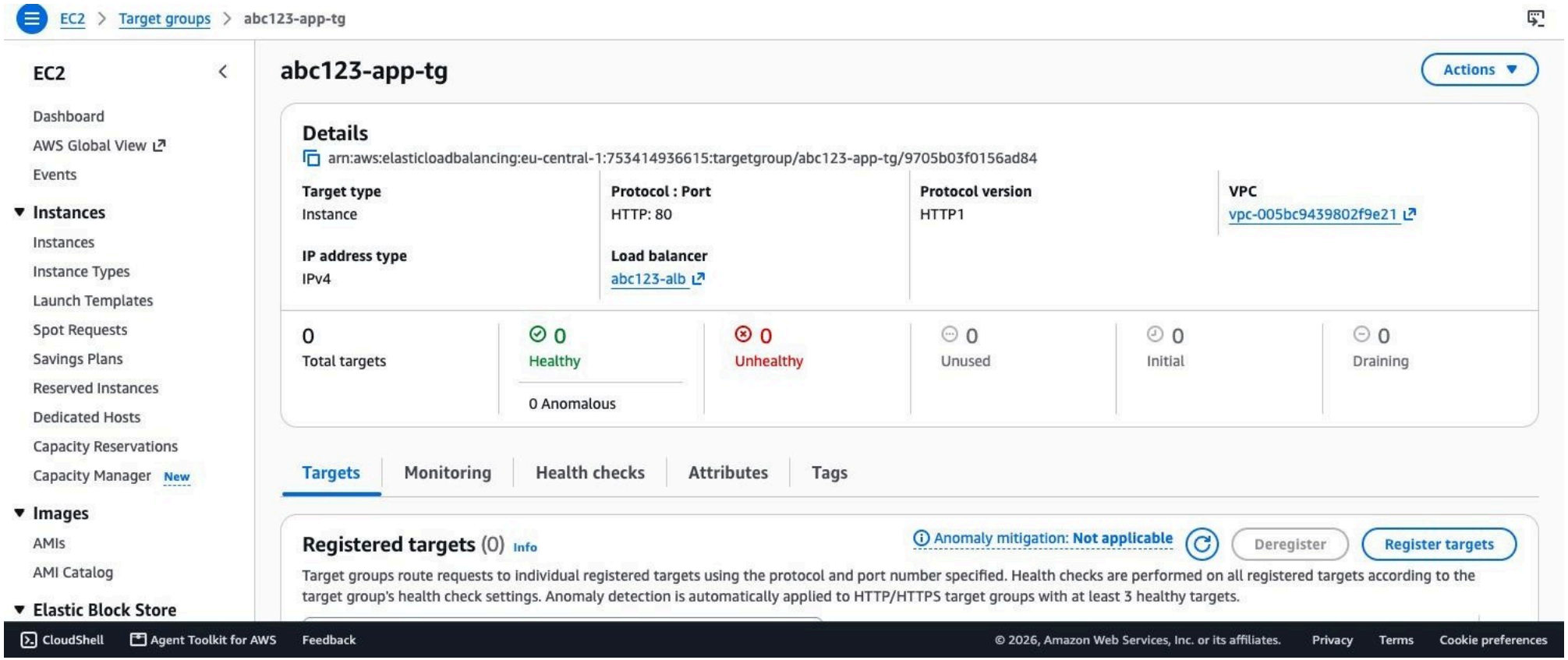
lb_address = "abc123-alb-2060495477.eu-central-1.elb.amazonaws.com"
```

Two minutes; the load balancer is the slow one. The first **output**: a DNS name, not an address, because the load balancer answers from one address per zone. Try it:

```
curl -i http://abc123-alb-2060495477.eu-central-1.elb.amazonaws.com/hello
HTTP/1.1 503 Service Temporarily Unavailable
Server: awselb/2.0
```

Correct. A load balancer with an empty target group is up and has nobody to forward to. `terraform state list | wc -l` says 19.

Step 3 — in the console



EC2 → Target groups → `abc123-app-tg` : HTTP 80, attached to `abc123-alb` , zero targets. The instances join it in step 6, from their own file.

Step 3 — what you wrote

- `aws_lb.lb` : `load_balancer_type = "application"` , `subnets = aws_subnet.public[*].id` (the splat: every instance of the counted subnet, as a list), `security_groups = [aws_security_group.alb.id]`
- `aws_lb_target_group.app` : port 80, `vpc_id` , and a `health_check { path = "/hello" }` block: the probe
- `aws_lb_listener.http` : `load_balancer_arn = aws_lb.lb.arn` , port 80, `default_action { type = "forward", target_group_arn = ... }` . Three blocks, two references, one build order
- `output "lb_address" { value = aws_lb.lb.dns_name }` : an **attribute**, assigned by the cloud, printed after every apply. The name you give the validator

Step 4 — the way out: plan

```
cp steps/04-way-out.tf .
terraform plan -var-file=me.tfvars
```

```
  # aws_eip.nat[0] will be created
  # aws_nat_gateway.nat[0] will be created
  # aws_route.private_default[0] will be created
  # aws_route_table.private will be created
  # aws_route_table_association.app will be created
  # aws_route_table_association.db[0] will be created
  # aws_route_table_association.db[1] will be created

Plan: 7 to add, 0 to change, 0 to destroy.
```

Week 2 gave each instance a temporary public IP so that it could install Docker. This configuration never does that: the **app subnet** gets a route through a **NAT gateway** in the first public subnet, one elastic IP, shared, outbound only.

Three names end in `[0]`: the address, the gateway and the default route have a `count`. The route table and its three associations exist either way.

Step 4 — the way out: apply

```
terraform apply -var-file=me.tfvars
  Enter a value: yes

aws_route_table.private: Creation complete after 0s [id=rtb-0c36336329ca041fb]
aws_route_table_association.app: Creation complete after 1s [id=rtbassoc-0b9565c262c0d264c]
aws_route_table_association.db[0]: Creation complete after 1s [id=rtbassoc-05459e7c5dd6d35b9]
aws_route_table_association.db[1]: Creation complete after 1s [id=rtbassoc-067e3b5fc9430a033]
aws_eip.nat[0]: Creation complete after 1s [id=eipalloc-087ad5d10784c4d1c]
aws_nat_gateway.nat[0]: Creation complete after 1m55s [id=nat-07c2ec31960d18b2d]
aws_route.private_default[0]: Creation complete after 0s [id=r-rtb-0c36336329ca041fb1080289494]

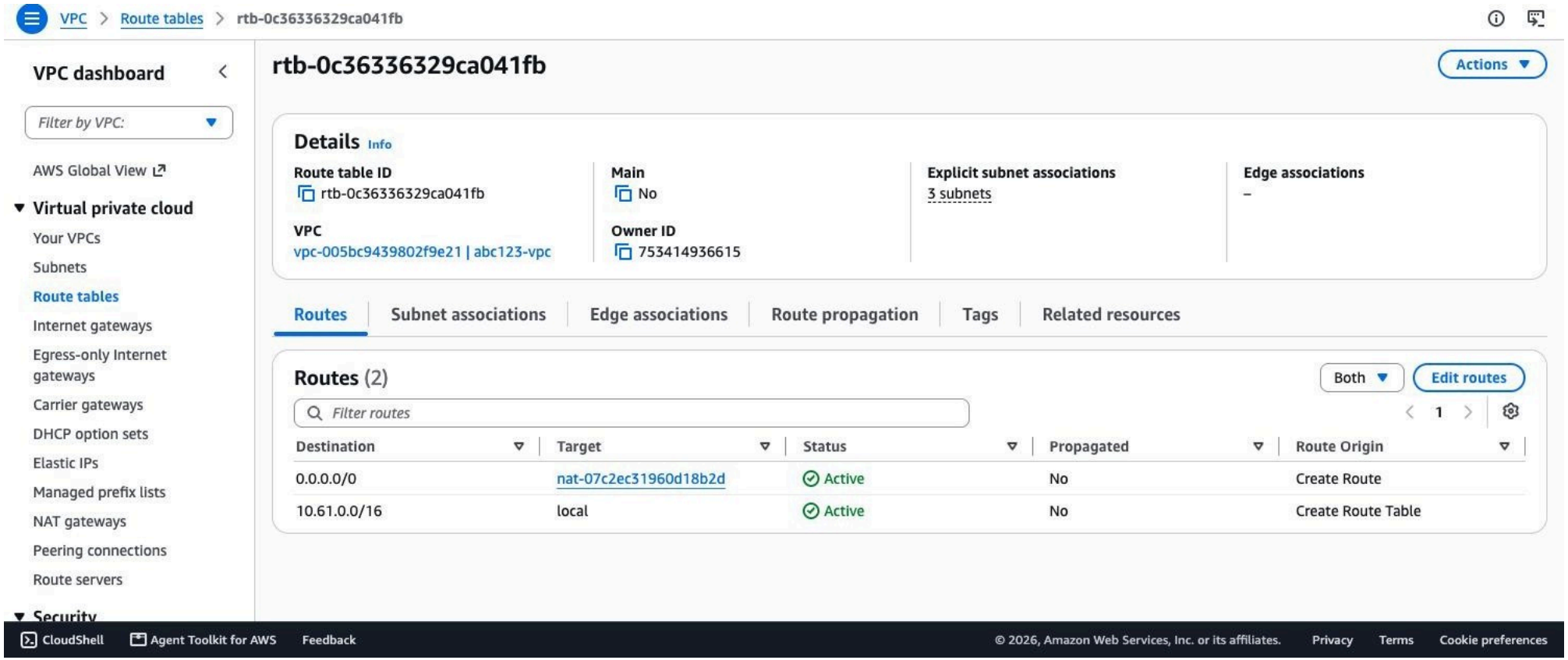
Apply complete! Resources: 7 added, 0 changed, 0 destroyed.

Outputs:

lb_address = "abc123-alb-2060495477.eu-central-1.elb.amazonaws.com"
locked = false
```

- Two minutes, nearly all of it the gateway. The route is created last: it needs the gateway's id
- A second output, `locked`, so that the last line of every apply says which phase you are in. `terraform state list | wc -l` says 26

Step 4 — in the console



VPC → *Route tables* → yours, not *Main* → *Routes*: `0.0.0.0/0` → `nat-…`, the VPC's range → `local`. *Subnet associations*: `app`, `db-0`, `db-1`. *NAT gateways* → `abc123-nat`: *Available*, in `public-0`.

Step 4 — what you wrote

```
resource "aws_nat_gateway" "nat" {
  count = var.locked ? 0 : 1
  ...
}
```

- `aws_eip.nat` and `aws_nat_gateway.nat`, both counted; the gateway sits in `aws_subnet.public[0]` and uses `aws_eip.nat[0].id`. References to a counted resource carry the index
- `aws_route_table.private` with no routes of its own, and `aws_route.private_default`, counted, pointing `0.0.0.0/0` at `aws_nat_gateway.nat[0].id`. The route is its own resource so that the lock-down destroys it cleanly; a route written inside the table block would outlive the gateway as a dead entry
- Three associations: the app subnet and both db subnets use the private table from now on. Nothing in the db subnets ever uses the route; RDS needs no way out
- `output "locked"` prints the variable after every apply

Step 5 — the database: plan

```
cp steps/05-database.tf .
terraform plan -var-file=me.tfvars
```

```
  # aws_db_instance.db will be created
  # aws_db_subnet_group.db will be created

Plan: 2 to add, 0 to change, 0 to destroy.
```

RDS for PostgreSQL: you ask for a version and a size, AWS runs it, patches it, backs it up. Two blocks: a **DB subnet group** listing the two db subnets, because RDS requires subnets in two zones, and the **instance** in that group, with the `db` security group, not publicly accessible.

Creation takes **six minutes**. Start the apply, then open `steps/06-app-vms.tf` and read it while the instance builds.

Step 5 — the database: apply

```
terraform apply -var-file=me.tfvars
  Enter a value: yes

aws_db_subnet_group.db: Creation complete after 1s [id=abc123-db-subnets]
aws_db_instance.db: Still creating... [10s elapsed]
...
aws_db_instance.db: Still creating... [6m0s elapsed]
aws_db_instance.db: Creation complete after 6m7s [id=db-CV4DKKV5OUT26YLQHSBC2VUA3Q]

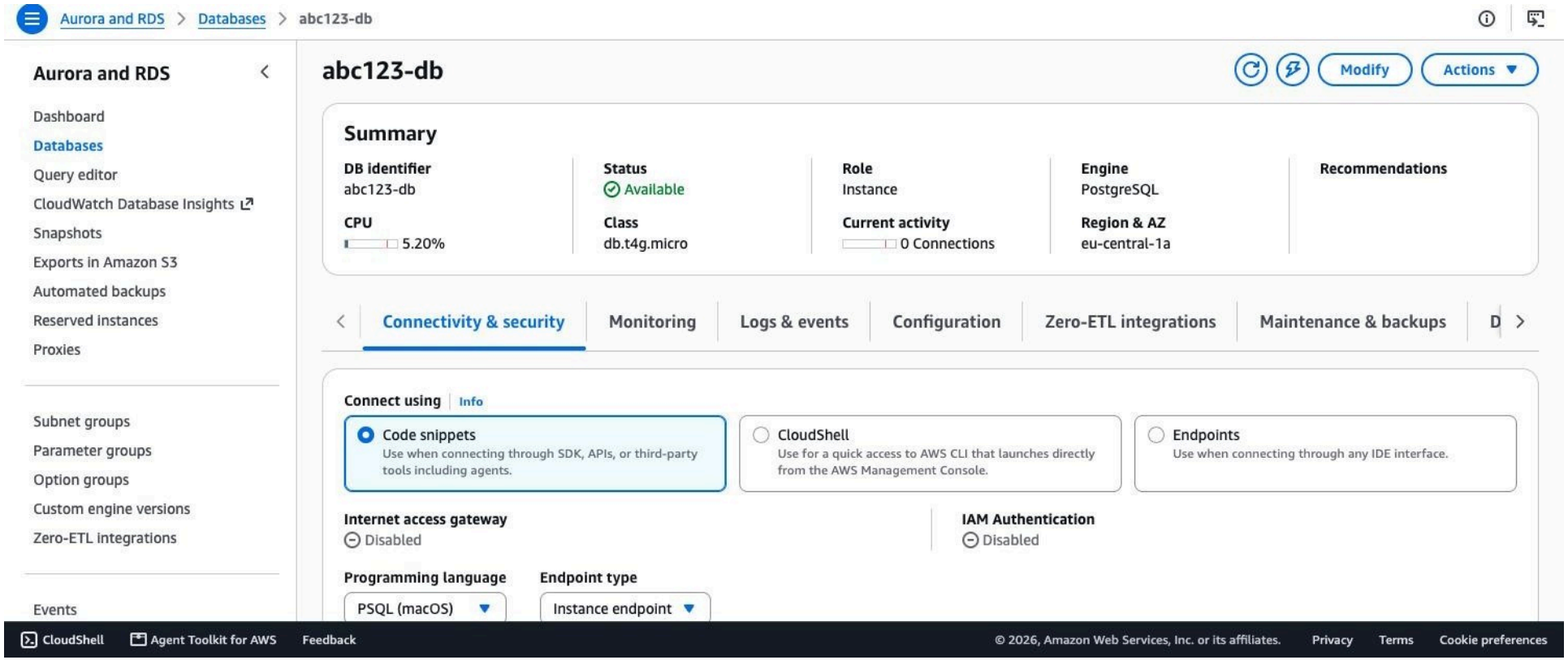
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

db_host = "abc123-db.cllw9hzvocae.eu-central-1.rds.amazonaws.com"
lb_address = "abc123-alb-2060495477.eu-central-1.elb.amazonaws.com"
locked = false
```

`db_host` is the instance's hostname. Nobody will type it anywhere: step 6 reads it from here. From your laptop the name resolves to `10.61.20.76`, a private address nothing outside the VPC can reach. `terraform state list | wc -l` says 28.

Step 5 — in the console



Aurora and RDS → *Databases* → `abc123-db` : *Available*, `db.t4g.micro` , zone `eu-central-1a` .
Further down on *Connectivity & security*: publicly accessible **No**, subnet group and security group yours.

Step 5 — what you wrote

- `aws_db_subnet_group.db`: `subnet_ids = aws_subnet.db[*].id`, the splat again
- `aws_db_instance.db`: `engine = "postgres"`, `engine_version = "16"`, `instance_class = "db.t4g.micro"`, `allocated_storage = 20`, `username = "beacon"`, `password = var.db_password`, `vpc_security_group_ids = [aws_security_group.db.id]`, `publicly_accessible = false`
- `skip_final_snapshot = true`: without it `destroy` stops to take a backup and asks for a name. Right for a lab, wrong for anything with data in it
- `output "db_host" { value = aws_db_instance.db.address }`

Step 6 — the app instances: plan

```
cp steps/06-app-vms.tf .
terraform plan -var-file=me.tfvars
```

```
  # aws_instance.app[0] will be created
  # aws_instance.app[1] will be created
  # aws_key_pair.admin will be created
  # aws_lb_target_group_attachment.app[0] will be created
  # aws_lb_target_group_attachment.app[1] will be created

Plan: 5 to add, 0 to change, 0 to destroy.
```

Week 2's two instances, with three differences: **no public IP**, **no SSH session**, and the beacon started by the instance itself at first boot. Plus a **key pair** made from your public key, used by these instances and by the bastion. Two blocks with `count = 2`: `app[0]` and `app[1]` of each.

Step 6 — the app instances: apply

```
terraform apply -var-file=me.tfvars
  Enter a value: yes

aws_key_pair.admin: Creation complete after 0s [id=abc123-key]
aws_instance.app[1]: Creation complete after 13s [id=i-03f0f5a67aef55ec4]
aws_instance.app[0]: Creation complete after 13s [id=i-019e9ac09cebc2736]
aws_lb_target_group_attachment.app[0]: Creation complete after 1s [id=…,i-019e9ac09cebc2736,80]
aws_lb_target_group_attachment.app[1]: Creation complete after 1s [id=…,i-03f0f5a67aef55ec4,80]

Apply complete! Resources: 5 added, 0 changed, 0 destroyed.
```

Twenty seconds. Then about a minute while the boot script installs Docker and pulls the image through the NAT gateway, and the load balancer's health checks turn green. `terraform state list | wc -l` says 33.

```
curl http://abc123-alb-2060495477.eu-central-1.elb.amazonaws.com/hello
```

The first tries answer 503 like in step 3; then the beacon answers.

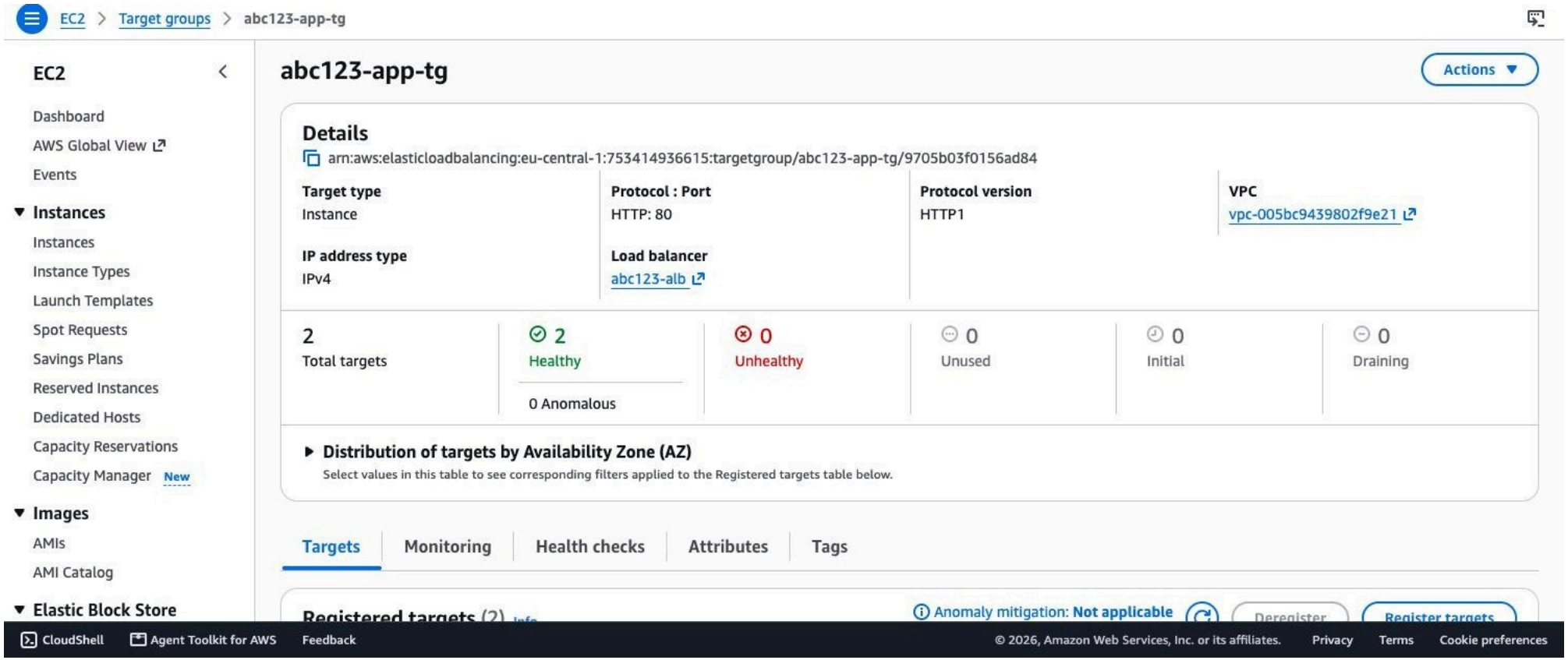
Step 6 — what the beacon says

```
{ "beacon_id": "b0c4538bb06f", "hostname": "ip-10-61-10-11",
  "interfaces": [ { "name": "ens5", "ips": ["10.61.10.11/24", ...] }, ... ],
  "outbound": [ { "target": "1.1.1.1:443",     "ok": true },
                { "target": "8.8.8.8:53",      "ok": true },
                { "target": "dns:example.com", "ok": true } ],
  "db_check": { "target": "abc123-db.cllw9hzvocae.eu-central-1.rds.amazonaws.com:5432",
                "ok": true, "addr": "10.61.20.76" },
  "cloud": { "provider": "aws", "vm_id": "i-03f0f5a67aef55ec4", "region": "eu-central-1" } }
```

- `outbound` : the two TCP probes **succeed**. Expected: the NAT gateway is up, this is the build phase
- `db_check` : `ok: true` , `addr: 10.61.20.76` . The beacon resolved the database's name to an address in a `db` subnet and connected. Nobody typed that name anywhere
- `interfaces` : `10.61.10.11` . No public one. The hostname is AWS's default, the address with dashes

Run it six times: `ip-10-61-10-10` , `ip-10-61-10-11` , `ip-10-61-10-10` , … Two machines, one name.

Step 6 — in the console



EC2 → Target groups → `abc123-app-tg` : 2 targets, 2 healthy. *Instances*: `abc123-app-1` and `abc123-app-2` , *Running*, and their *Public IPv4 DNS* column is empty.

Step 6 — what you wrote

- `aws_key_pair.admin` : `public_key = var.admin_ssh_key` . Week 2's *Create key pair* dialog, with your own key
- `aws_instance.app` : `count = 2` , `ami = data.aws_ami.ubuntu.id` from step 1's lookup, `private_ip = cidrhost("10.61.10.0/24", 10 + count.index)` : `.10` and `.11` . No `associate_public_ip_address`
- `user_data = templatefile("user-data-app.sh", { beacon_image = var.beacon_image, db_target = "${aws_db_instance.db.address}:5432" })` : the boot file with its holes filled. That reference is why the instances are created **after** the database
- `aws_lb_target_group_attachment.app` : `count = 2` , `target_id = aws_instance.app[count.index].id` . Step 3's group, filled
- `user-data-app.sh` : install Docker, `docker run ... -e DB_TARGET=${db_target} ${beacon_image}` . What you typed over SSH last week, done by the instance at boot

Step 7 — the bastion: plan

```
cp steps/07-bastion.tf .
terraform plan -var-file=me.tfvars
```

```
  # aws_instance.bastion will be created

Plan: 1 to add, 0 to change, 0 to destroy.
```

The app instances never had a public IP, so the way in is one small instance in the first public subnet, with the second and last public address:

```
laptop ──22, from your IP only──▶ bastion ──22, from the bastion group only──▶ app instances
```

Both rules exist since step 2. The bastion runs nothing but SSH: no Docker, no application.

Step 7 — the bastion: apply

```
terraform apply -var-file=me.tfvars
  Enter a value: yes

aws_instance.bastion: Creation complete after 13s [id=i-089ca24d813ac710a]

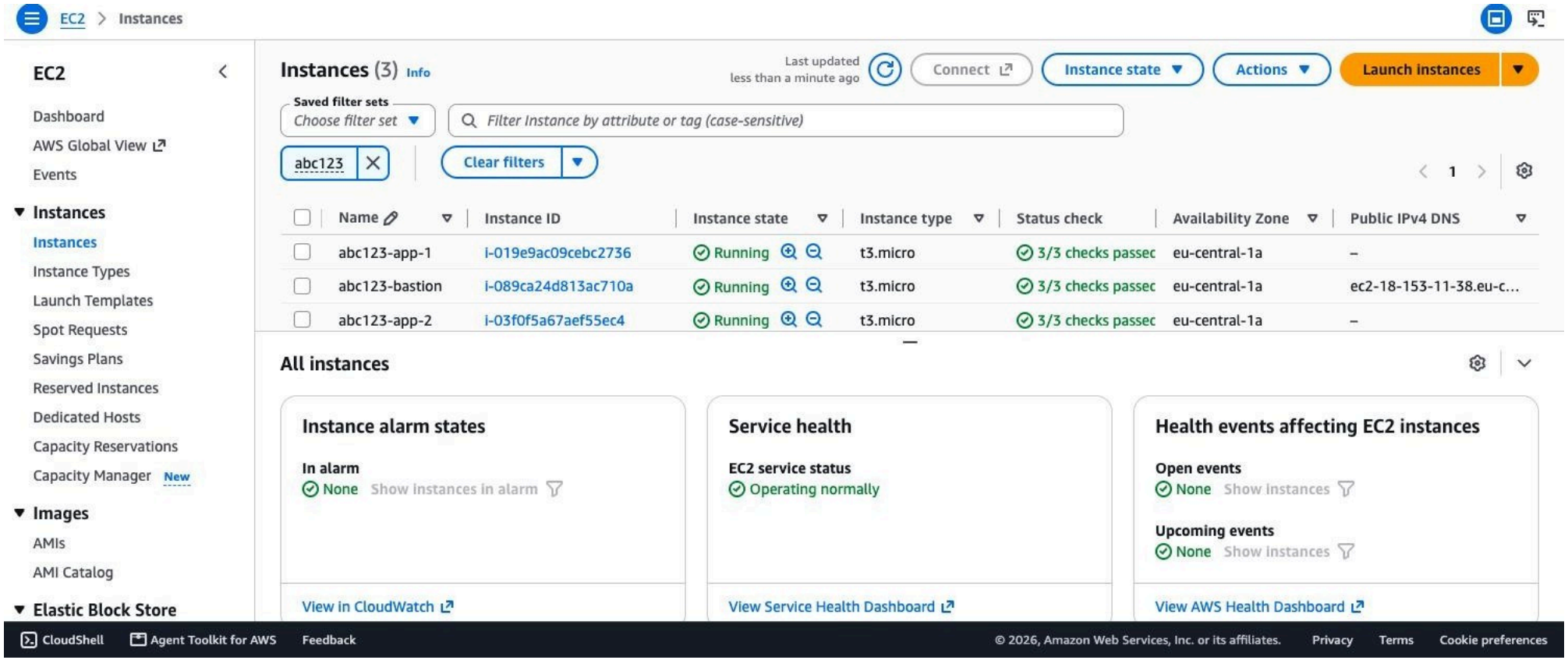
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Outputs:

app_private_ips = [ "10.61.10.10", "10.61.10.11" ]
bastion_address = "18.153.11.38"
db_host = "abc123-db.cllw9hzvocae.eu-central-1.rds.amazonaws.com"
lb_address = "abc123-alb-2060495477.eu-central-1.elb.amazonaws.com"
locked = false
ssh_command = "ssh -J ubuntu@18.153.11.38 ubuntu@10.61.10.10"
```

`terraform state list | wc -l` says **34**: 32 resources and 2 data sources. That is the whole assignment; from here on nothing is added. `ssh_command` was assembled by the configuration from two attributes: the bastion's public IP and app-1's private IP.

Step 7 — in the console



EC2 → Instances, filtered by `abc123`: three instances, all *Running*, all in zone `a`. Only `abc123-bastion` has a *Public IPv4 DNS*.

Step 7 — look inside through the bastion

```
ssh -J ubuntu@18.153.11.38 ubuntu@10.61.10.10
```

`-J` jumps through the bastion to the private address; your key stays on the laptop. Inside app-1:

```
ubuntu@ip-10-61-10-10:~$ sudo docker ps
NAMES     IMAGE                                  STATUS
beacon    ghcr.io/siimrebane/iti8801-beacon:a3   Up About a minute
ubuntu@ip-10-61-10-10:~$ nc -zv -w 5 abc123-db.cllw9hzvocae.eu-central-1.rds.amazonaws.com 5432
Connection to abc123-db.cllw9hzvocae.eu-central-1.rds.amazonaws.com (10.61.20.76) 5432 port [tcp/postgresql] succeeded!
ubuntu@ip-10-61-10-10:~$ exit
```

Now the same test from the **bastion** itself:

```
ssh ubuntu@18.153.11.38
ubuntu@ip-10-61-0-230:~$ nc -zv -w 5 abc123-db.cllw9hzvocae.eu-central-1.rds.amazonaws.com 5432
nc: connect to abc123-db.cllw9hzvocae.eu-central-1.rds.amazonaws.com (10.61.20.76) port 5432 (tcp) timed out
```

Same VPC, same database, different group: the `db` group names `app` and not `bastion`. That failure is the database tier working as drawn.

Step 7 — what you wrote

- `aws_instance.bastion` : same AMI, same type, same key pair as the app instances; `subnet_id = aws_subnet.public[0].id` , `associate_public_ip_address = true` , the bastion group, and no `user_data` : nothing to install
- `output "bastion_address"` : its `public_ip` . `output "ssh_command"` : a string built from two attributes. The command you type is assembled by the configuration

Step 8 — lock down: plan

```
terraform plan -var-file=me.tfvars -var locked=true
```

```
  # aws_eip.nat[0] will be destroyed
  # (because index [0] is out of range for count)
  # aws_nat_gateway.nat[0] will be destroyed
  # aws_route.private_default[0] will be destroyed

Plan: 0 to add, 0 to change, 3 to destroy.
```

Step 4's gateway, its address and the default route, one variable. Nothing else: no instance is touched, the database is not touched. If the plan mentions `aws_instance` or `aws_db_instance`, stop and ask.

Step 8 — lock down: apply

```
terraform apply -var-file=me.tfvars -var locked=true
  Enter a value: yes

aws_route.private_default[0]: Destruction complete after 0s
aws_nat_gateway.nat[0]: Destruction complete after 1m11s
aws_eip.nat[0]: Destruction complete after 1s

Apply complete! Resources: 0 added, 0 changed, 3 destroyed.

Outputs:
...
locked = true
```

A minute and a half; the gateway takes its time going as well as coming. The app subnet has no route out any more. `terraform state list | wc -l` says 31.

If the apply ends with an error about releasing the address, AWS was still tearing the gateway down when the address was released. Run the same apply again; it finishes the one thing left.

Step 8 — verify the lock-down

Wait a minute for the beacon's next probe round, then:

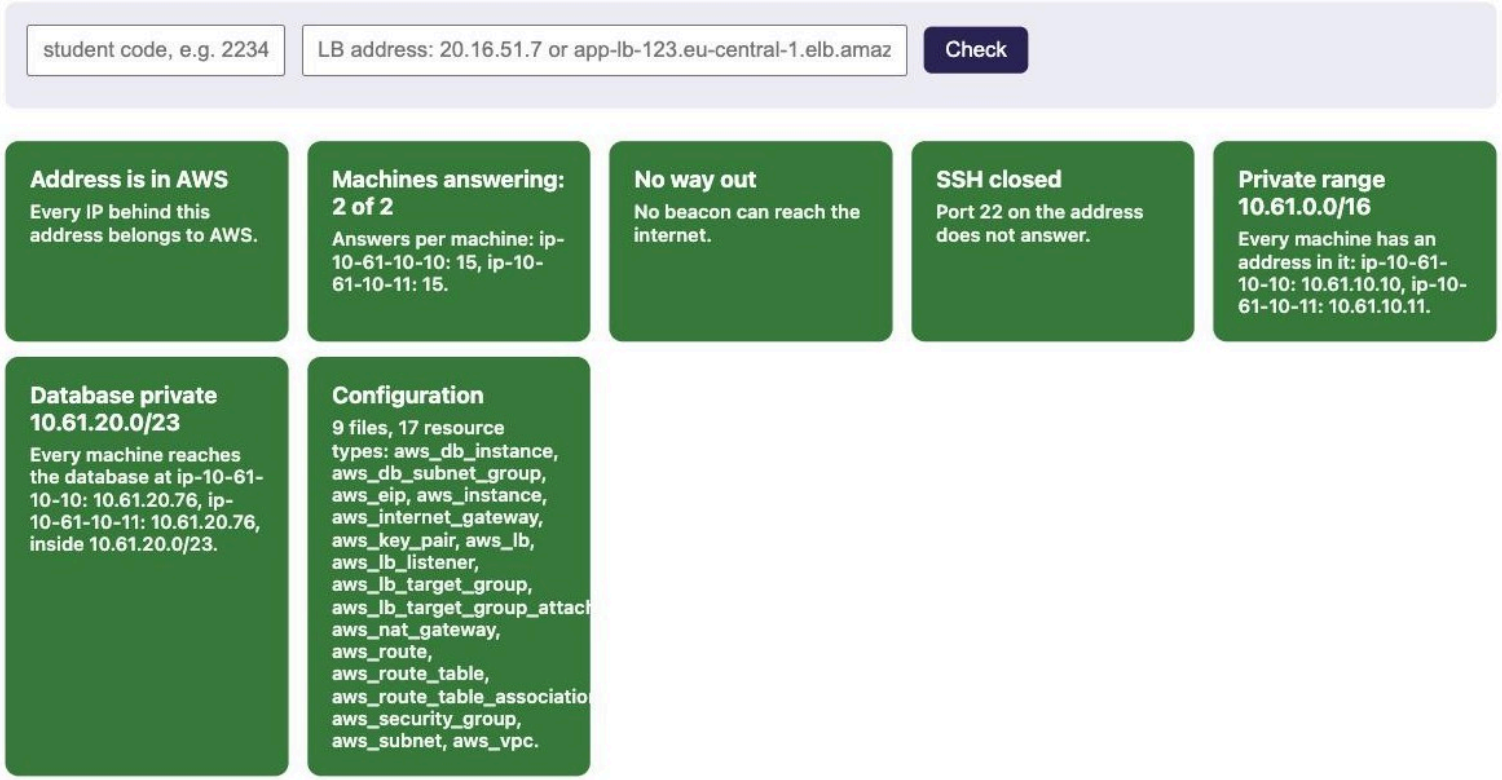
```
curl http://abc123-alb-2060495477.eu-central-1.elb.amazonaws.com/hello
```

```
  "outbound": [ { "target": "1.1.1.1:443",     "ok": false },
                { "target": "8.8.8.8:53",      "ok": false },
                { "target": "dns:example.com", "ok": true  } ],
  "db_check": { "target": "abc123-db.cllw9hzvocae.eu-central-1.rds.amazonaws.com:5432",
                "ok": true, "addr": "10.61.20.76" },
```

- Both TCP probes **fail** with a timeout. The app subnet has no way out
- `dns:example.com` still succeeds: the VPC's resolver is a platform service at the VPC's `.2` address, not the internet. The validator ignores it
- `db_check` still `ok`. Private traffic does not need the internet
- The load balancer still answers: traffic between it and the instances stays inside the VPC. The bastion still works: its subnet routes through the internet gateway, not the NAT gateway

Step 9 — the check

https://iti8801.digivader.com/lab3-validator/ · student code, the load balancer's DNS name (`lb_address`, nothing else: no `http://`, no port), *Check*. Thirty requests through your load balancer, about a minute:



Step 9 — the last step: your configuration

Six green boxes; the seventh, *Configuration*, is grey until you give it the files. Under the check code: *Upload configuration*. Select every file of `terraform/aws` except `me.tfvars` and the state: the nine `.tf` files, `user-data-app.sh`, `.gitignore`, `.terraform.lock.hcl`.

The validator checks three things, and stores nothing until all three hold:

- the files are Terraform: blocks and braces, nothing else at the top level
- the resource types of this lab are in them: instances, load balancer, database, NAT gateway, security groups, and the `locked` variable
- no `tfvars`, no state, no key came along. A refused upload names the file; remove it and upload again

Step 9 — passed



Seven green boxes and this receipt. Copy the code, paste it into Moodle. The check and the files are stored on the server; nothing you delete afterwards changes it. Every check gets its own code; submit a passed one.

Step 9 — not green?

Red box	Usually
Machines answering: 1 of 2	one target is not healthy yet; wait two minutes, or `docker logs beacon` through the bastion
No way out	`locked=true` not applied, or the plan you applied did not destroy the NAT gateway
SSH closed	cannot go red with this configuration: the listener forwards port 80 only
Database private	`db_check` not ok: the instances booted before the database answered, or the beacon tag is not `a3`
Database private, address outside the range	the instance is not in the db subnet group; the plan will show why
Configuration	a `tfvars` or state file in the selection, or a file that is not Terraform; the message names it

Fix in the file, `plan`, `apply`, check again. Every check gets its own code; submit a green one. **Never** fix by clicking in the console: the next `plan` shows the drift.

Step 9 — destroy

```
terraform destroy -var-file=me.tfvars -var locked=true
```

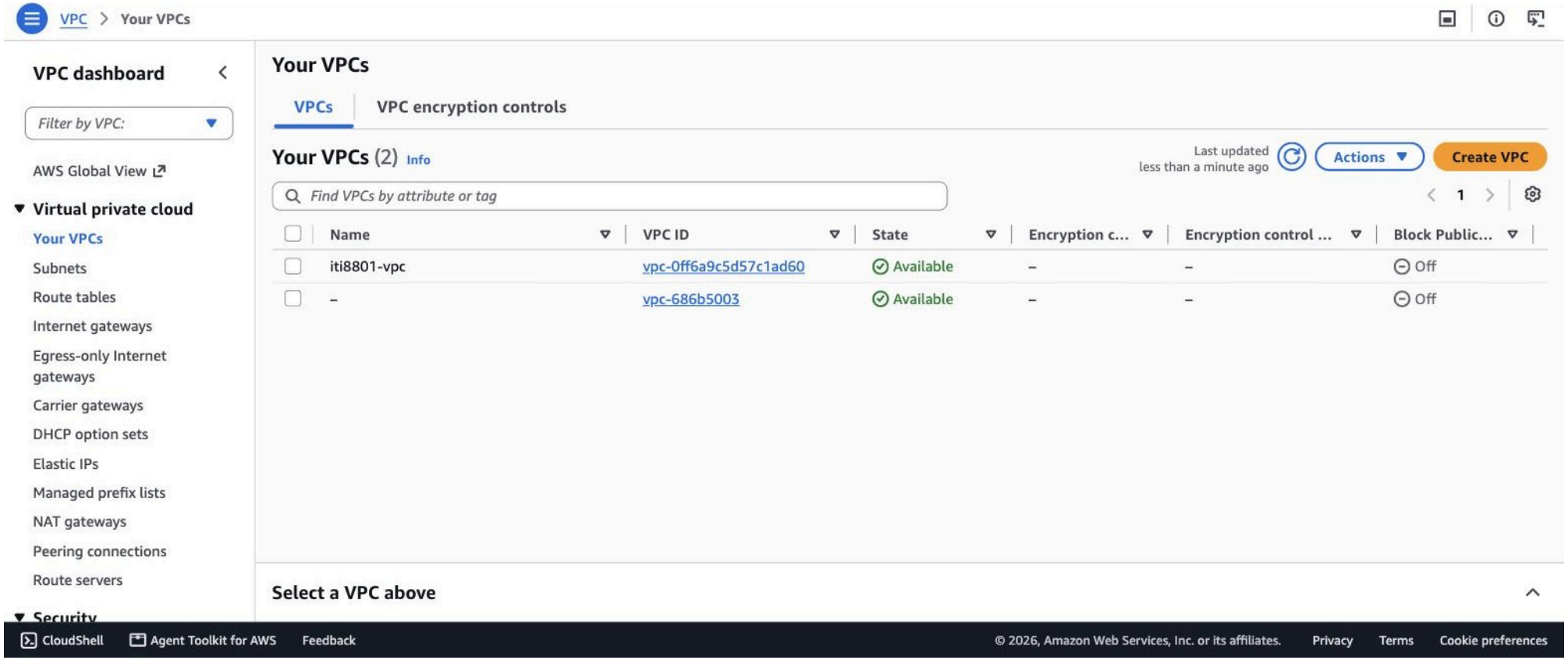
```
Plan: 0 to add, 0 to change, 29 to destroy.
  Enter a value: yes

aws_lb_target_group_attachment.app[1]: Destruction complete after 0s
...
aws_lb.lb: Destruction complete after 17s
aws_instance.bastion: Destruction complete after 30s
aws_instance.app[0]: Destruction complete after 30s
aws_db_instance.db: Destruction complete after 1m52s
aws_vpc.vpc: Destruction complete after 1s

Destroy complete! Resources: 29 destroyed.
```

Everything in the state, in reverse order, two and a half minutes; the database takes most of it. Save this output too. Same `locked=true` on destroy as on the last apply: otherwise the plan first **creates** the NAT gateway to destroy it a moment later.

Step 9 — in the console



Your VPCs: none of yours, the default stays. *Elastic IPs*: none of yours. *Databases*: none. Anything of yours still there was created outside Terraform; the console is the only way to remove it.

If something fails

- `apply` stops with an error: read it, it names the resource and the reason. Fix the cause, `apply` again; it continues from where it stopped
- `AddressLimitExceeded`: the elastic IP quota. Release unused addresses in *EC2 → Elastic IPs*, `apply` again
- `InsufficientDBInstanceCapacity` or an unsupported class: ask; the answer depends on the region, and it goes into the file, not the console
- The beacon never answers: the instances booted before the database existed, or the image pull failed. `terraform apply -replace=aws_instance.app[0] -var-file=me.tfvars` rebuilds one instance
- Truly stuck: `terraform destroy` and start over. Rebuilding is seven commands and about fifteen minutes, or one command with all seven files in place

A3 — what you submit

- The validator check code, from a check that is green through the seventh box, in Moodle
- The configuration goes to the validator at that seventh box, not to Moodle: all seven step files, `providers.tf`, `variables.tf`, the boot file, `.gitignore`, the lock file. **No `terraform.tfstate`, no `me.tfvars`**: the validator refuses them and stores nothing until they are gone
- The full output of `terraform apply` from nothing, and of `terraform destroy`. Seven applies today; for the submission one apply with all seven files is fine, and it is the same 32 resources

Deadline: **Sunday of week 4, 23:59.**

Done when

- Seven files, seven plans read, seven applies, and after each one you saw the new thing and a `plan` that said *No changes*
- You saw the database's hostname arrive in the beacon's report without typing it anywhere
- You reached an app instance through the bastion, and saw the database refuse the bastion
- `locked=true` removed the NAT gateway and its address, and the beacon's probes failed afterwards
- The assignment 3 validator is green, all seven boxes, and its code is in Moodle
- `terraform destroy` ran, and the console shows no VPC of yours, no elastic IP, no database



Questions