# ITI8801 · Practice 3 · Azure

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

Siim Rebane · TalTech · Autumn 2026

# The task

Build, on your own Azure subscription, with Terraform and nothing else:

- the **network** of week 2: a VNet, an app subnet with two app VMs running the course beacon, a load balancer as the only door
- a **managed PostgreSQL** server in a second private subnet, 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 VM ever has a public IP

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

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` | resource group, VNet, three subnets | 5 to add |
| 2 | `02-walls.tf` | three NSGs, attached to the subnets | 6 |
| 3 | `03-load-balancer.tf` | public IP, load balancer, pool, probe, rule | 5 |
| 4 | `04-way-out.tf` | NAT gateway and its public IP, while `locked=false` | 4 |
| 5 | `05-database.tf` | private DNS zone, link, PostgreSQL server | 3 |
| 6 | `06-app-vms.tf` | two NICs, two VMs, pool membership | 6 |
| 7 | `07-bastion.tf` | public IP, NIC, bastion VM | 3 |

# Final state, and how it is proven

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

1. the address belongs to Azure
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 `10.60.0.0/16`
6. every beacon reaches the database, and the database's address is in **`10.60.2.0/24`**

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, `az 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, five minutes, spent reading step 6 | 10 min |
| 6 | the app VMs; 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 subscription | 15 min |

# Before you start: the public IP quota

Azure for Students allows **three** public IP addresses. This configuration uses all three:

| Public IP | Belongs to | Exists |
|---|---|---|
| 1 | the load balancer | always |
| 2 | the NAT gateway | until `locked=true` |
| 3 | the bastion | always |

So **nothing from week 1 or week 2 may still exist** in the subscription. Portal → *Resource groups*: if anything is there, delete it now, and wait until the deletion has finished before the first `apply`. A1 was graded from the running VM and A2 from the stored check; nothing needs to survive.

Cost: the NAT gateway, the database and the three VMs are billed by the hour from your credit, about 15 cents an hour together. Today's build lives about two hours; a month unattended would be most of the credit. `destroy` at the end is not optional.

# Phase 0 — tools

```
terraform version        # 1.6 or newer; OpenTofu works the same, the command is tofu
az login                 # opens a browser; log in with the account of your student subscription
az account show          # name: Azure for Students; copy the id
```

`az account show` prints three ids. The subscription id is the one called `id`; the two tenant ids are the university's, not yours:

```
{
  "homeTenantId": "3efd4d88-...",                  # not this
  "id": "5f1d33b3-1af8-40fe-a852-012c379cd107",    # this one: your subscription id
  "name": "Azure for Students",
  "tenantId": "3efd4d88-...",                      # not this either
  ...
}
```

The portal shows the same value under *Subscriptions → Azure for Students → Subscription ID*.

# Phase 0 — the configuration

Get the configuration from **github.com/siimrebane/iti-8801**, then:

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

`init` downloads the azurerm provider, about 200 MB. Start it now; it runs while we read.

# Phase 0 — providers.tf

Open `providers.tf` while `init` runs. Two lines matter:

```
subscription_id                 = var.subscription_id
resource_providers_to_register  = ["Microsoft.DBforPostgreSQL"]
```

The first: the provider refuses to run without a subscription id. Yours goes into `me.tfvars`.

The second: a subscription can create resources of a service only after that service is registered on it, once. The portal did that silently for VMs and networks in week 1. Nothing has registered PostgreSQL on your subscription yet, and Terraform does not go through the portal, so without this line step 5 would fail with `MissingSubscriptionRegistration`. Nothing to type: the line makes Terraform register it during your first `plan`.

# Phase 0 — your variables

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

```
subscription_id = "5f1d33b3-..."              # az account show --query id -o tsv
location        = "swedencentral"             # 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
```

- `vm_size` has a default; change it only if the plan says the size is not free 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
```

The first plan pauses before it prints anything: Azure is registering the PostgreSQL service on your subscription. That happens once, then never again. Then:

```
  # azurerm_resource_group.rg will be created
  # azurerm_subnet.app will be created
  # azurerm_subnet.db will be created
  # azurerm_subnet.public will be created
  # azurerm_virtual_network.vnet will be created

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

Five blocks in the file, five things to create: a resource group, a VNet `10.60.0.0/16`, and three subnets. Read the five, then:

# Step 1 — the network: apply

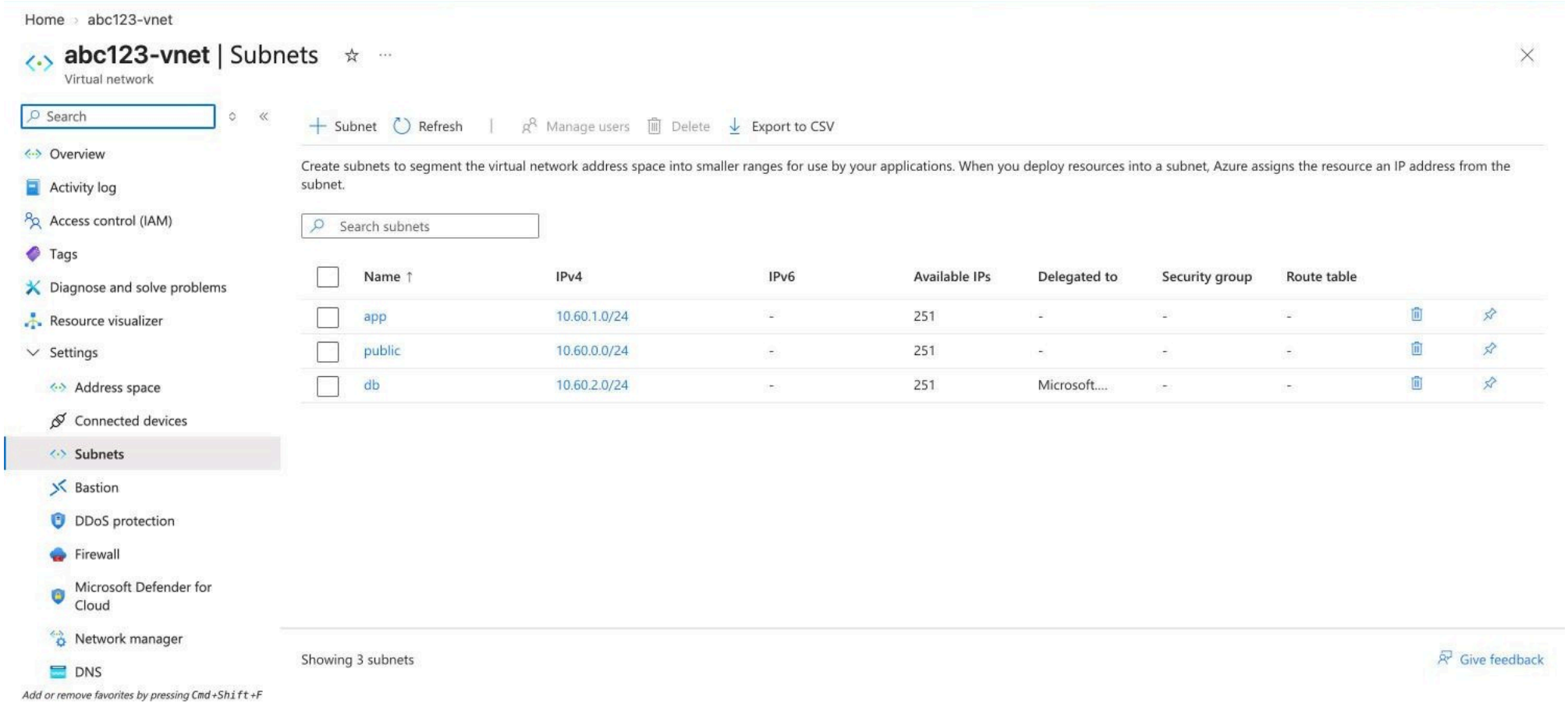
```
terraform apply -var-file=me.tfvars
  Enter a value: yes

azurerm_resource_group.rg: Creation complete after 24s [id=…/resourceGroups/abc123-rg]
azurerm_virtual_network.vnet: Creation complete after 6s [id=…/virtualNetworks/abc123-vnet]
azurerm_subnet.app: Creation complete after 6s [id=…/abc123-vnet/subnets/app]
azurerm_subnet.public: Creation complete after 12s [id=…/abc123-vnet/subnets/public]
azurerm_subnet.db: Creation complete after 18s [id=…/abc123-vnet/subnets/db]

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

- About a minute. The group first, then the VNet, then the three subnets at the same time: 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 five lines: what Terraform now knows it owns

# Step 1 — in the portal




*Resource groups* → `abc123-rg` → `abc123-vnet` → *Subnets*. Three subnets; `db` is delegated to PostgreSQL; no security group on any of them yet.

# Step 1 — what you wrote

| Subnet | Range | For |
|---|---|---|
| `public` | `10.60.0.0/24` | the bastion. The only subnet whose machine gets a public IP |
| `app` | `10.60.1.0/24` | the two app VMs; `default_outbound_access_enabled = false`, the "Private subnet" checkbox |
| `db` | `10.60.2.0/24` | the database; **delegated** to the PostgreSQL service, nothing else may live there |

- `name = "${var.prefix}-rg"`: every name starts with your prefix. One file serves the whole course
- `resource_group_name = azurerm_resource_group.rg.name` on the VNet, `virtual_network_name = azurerm_virtual_network.vnet.name` on each subnet: **references**. They carry the value and fix the order
- The `delegation { service_delegation { name = "Microsoft.DBforPostgreSQL/flexibleServers" } }` block on `db` is the portal's *Delegate subnet to a service* dropdown, as code

# Before every step: plan again

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

```
azurerm_resource_group.rg: Refreshing state... [id=…/resourceGroups/abc123-rg]
azurerm_virtual_network.vnet: Refreshing state... [id=…/virtualNetworks/abc123-vnet]
azurerm_subnet.app: Refreshing state... [id=…/abc123-vnet/subnets/app]
azurerm_subnet.public: Refreshing state... [id=…/abc123-vnet/subnets/public]
azurerm_subnet.db: Refreshing state... [id=…/abc123-vnet/subnets/db]

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

Nothing new in the files, so Terraform asked the cloud about its five 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
```

```
  # azurerm_network_security_group.app will be created
  # azurerm_network_security_group.bastion will be created
  # azurerm_network_security_group.db will be created
  # azurerm_subnet_network_security_group_association.app will be created
  # azurerm_subnet_network_security_group_association.bastion will be created
  # azurerm_subnet_network_security_group_association.db will be created

Plan: 6 to add, 0 to change, 0 to destroy.
```

Three network security groups, one per subnet, and the three attachments. Week 2 had one group with two rules, attached by hand.

# Step 2 — the walls: apply

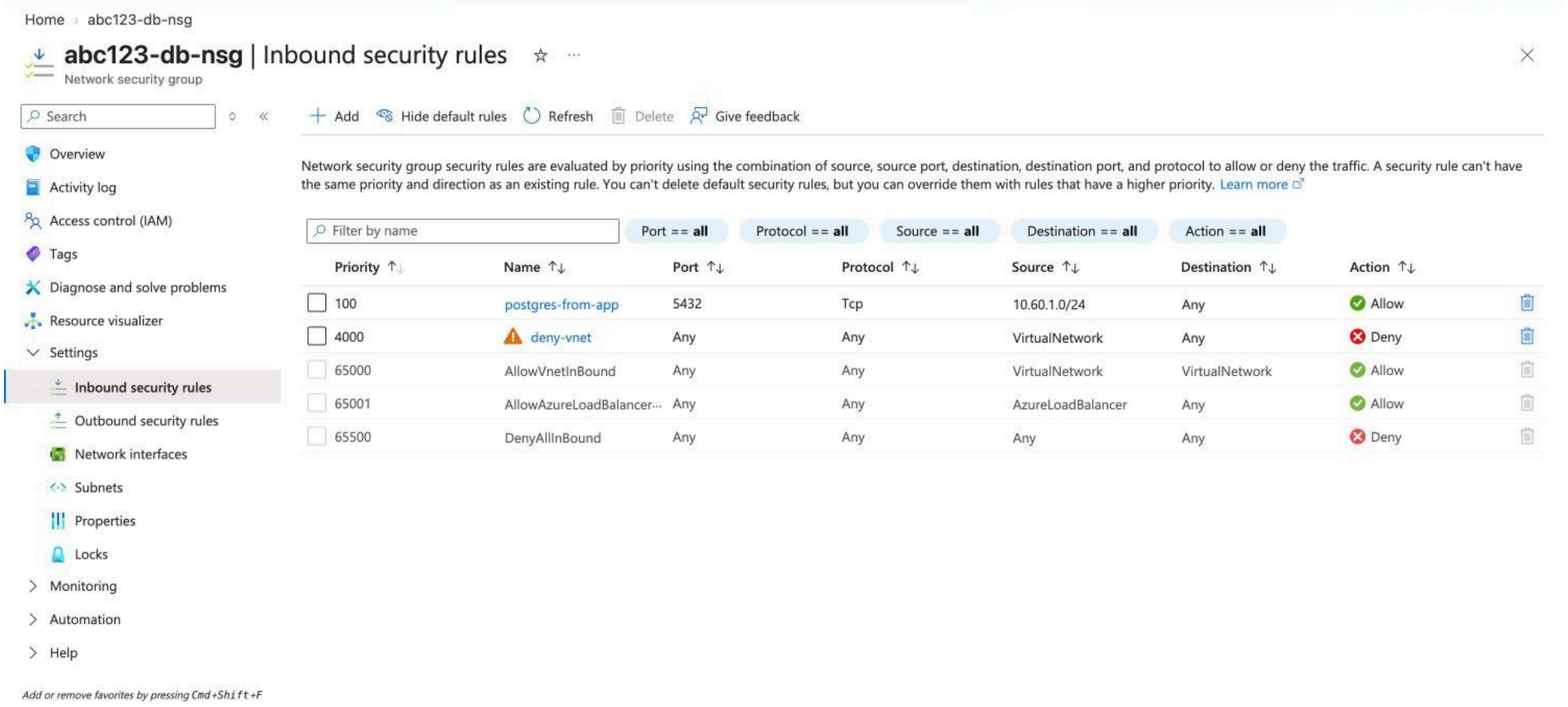
```
terraform apply -var-file=me.tfvars
  Enter a value: yes

azurerm_network_security_group.db: Creation complete after 3s [id=…/abc123-db-nsg]
azurerm_network_security_group.app: Creation complete after 4s [id=…/abc123-app-nsg]
azurerm_network_security_group.bastion: Creation complete after 4s [id=…/abc123-bastion-nsg]
azurerm_subnet_network_security_group_association.db: Creation complete after 5s [id=…/subnets/db]
azurerm_subnet_network_security_group_association.app: Creation complete after 10s [id=…/subnets/app]
azurerm_subnet_network_security_group_association.bastion: Creation complete after 15s [id=…/subnets/public]

Apply complete! Resources: 6 added, 0 changed, 0 destroyed.
```

- Forty seconds. The three groups at once, then the three attachments at once
- `terraform state list | wc -l` says 11. `terraform plan -var-file=me.tfvars`: *No changes*

# Step 2 — in the portal




*Network security groups* → `abc123-db-nsg` → *Inbound security rules*: your two rules above the three built-in ones, sorted by priority. The *Subnets* entry shows `db`.

# Step 2 — what you wrote

| NSG | Rule | Priority | Why |
|---|---|---|---|
| `app` | allow 80 from `Internet` | 100 | the load balancer delivers on port 80 |
| `app` | allow 22 from `10.60.0.0/24` | 110 | SSH only from the bastion's subnet |
| `db` | allow 5432 from `10.60.1.0/24` | 100 | PostgreSQL from the app subnet only |
| `db` | deny `*` from `VirtualNetwork` | 4000 | the built-in *AllowVnetInBound* would let every VM in the VNet in |
| `bastion` | allow 22 from `<your IP>/32` | 100 | one laptop, and no other address |

- Each `azurerm_network_security_group` holds its rules as `security_rule { ... }` blocks: the *Add inbound rule* form, field by field
- `source_address_prefix = "${var.my_ip}/32"` : your variable inside a string. New IP tomorrow: change one line, `apply` , *1 to change*
- Three `azurerm_subnet_network_security_group_association` blocks attach the walls; unattached, an NSG protects nothing

# Step 3 — the door: plan

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

```
  # azurerm_lb.lb will be created
  # azurerm_lb_backend_address_pool.app will be created
  # azurerm_lb_probe.http will be created
  # azurerm_lb_rule.http will be created
  # azurerm_public_ip.lb will be created

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

Week 2's load balancer, all five portal pages in one file: a public IP, the load balancer, an empty backend pool, a health probe on `/hello`, and the rule from frontend port 80 to backend port 80 through the probe.

# Step 3 — the door: apply

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

azurerm_public_ip.lb: Creation complete after 3s [id=…/publicIPAddresses/abc123-lb-ip]
azurerm_lb.lb: Creation complete after 11s [id=…/loadBalancers/abc123-lb]
azurerm_lb_probe.http: Creation complete after 12s [id=…/abc123-lb/probes/http-probe]
azurerm_lb_backend_address_pool.app: Creation complete after 14s [id=…/abc123-lb/backendAddressPools/app-pool]
azurerm_lb_rule.http: Creation complete after 4s [id=…/abc123-lb/loadBalancingRules/http]

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

Outputs:

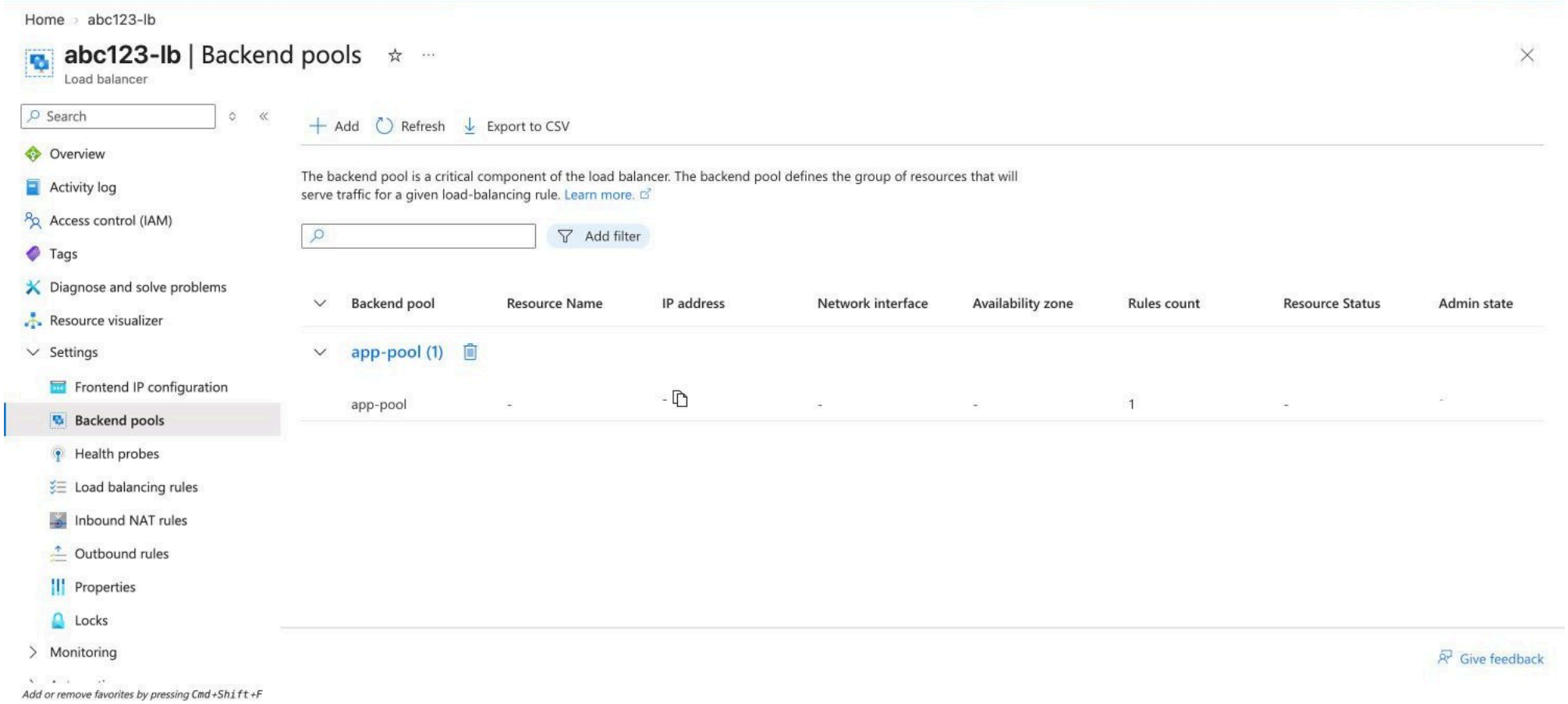
lb_address = "135.225.95.152"
```

The first **output**: the address the cloud assigned to the public IP. Try it:

```
curl -m 5 http://135.225.95.152/hello
curl: (28) Connection timed out after 5003 milliseconds
```

Correct. A load balancer with an empty pool has nobody to send the request to. `terraform state list | wc -l` says 16.

# Step 3 — in the portal




*Load balancers* → `abc123-lb` → *Backend pools*: `app-pool`, one rule, no instances. The VMs join it in step 6, from their own file.

# Step 3 — what you wrote

- `azurerm_public_ip.lb` : `sku = "Standard"` , `allocation_method = "Static"` . The first of the three public IPs
- `azurerm_lb.lb` references the IP by `azurerm_public_ip.lb.id` ; the pool, the probe and the rule reference the load balancer by `azurerm_lb.lb.id` . Five blocks, one chain of references, one build order
- `disable_outbound_snat = true` on the rule: week 2's *Use outbound rules* radio button. Without it the load balancer would be a way out for the VMs, and the validator would see it
- `output "lb_address" { value = azurerm_public_ip.lb.ip_address }` : an **attribute**, assigned by the cloud, printed after every apply. The address you give the validator

# Step 4 — the way out: plan

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

```
  # azurerm_nat_gateway.nat[0] will be created
  # azurerm_nat_gateway_public_ip_association.nat[0] will be created
  # azurerm_public_ip.nat[0] will be created
  # azurerm_subnet_nat_gateway_association.app[0] will be created

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

Week 2 gave each VM a temporary public IP so that it could install Docker. This configuration never does that: the **app subnet** gets a **NAT gateway**, one shared public IP, outbound only. Nothing can connect in through it.

Every name ends in `[0]`: these four blocks have a `count`.

# Step 4 — the way out: apply

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

azurerm_public_ip.nat[0]: Creation complete after 4s [id=…/publicIPAddresses/abc123-nat-ip]
azurerm_nat_gateway.nat[0]: Creation complete after 12s [id=…/natGateways/abc123-nat]
azurerm_subnet_nat_gateway_association.app[0]: Creation complete after 5s [id=…/subnets/app]
azurerm_nat_gateway_public_ip_association.nat[0]: Creation complete after 13s [id=…/abc123-nat|…/abc123-nat-ip]

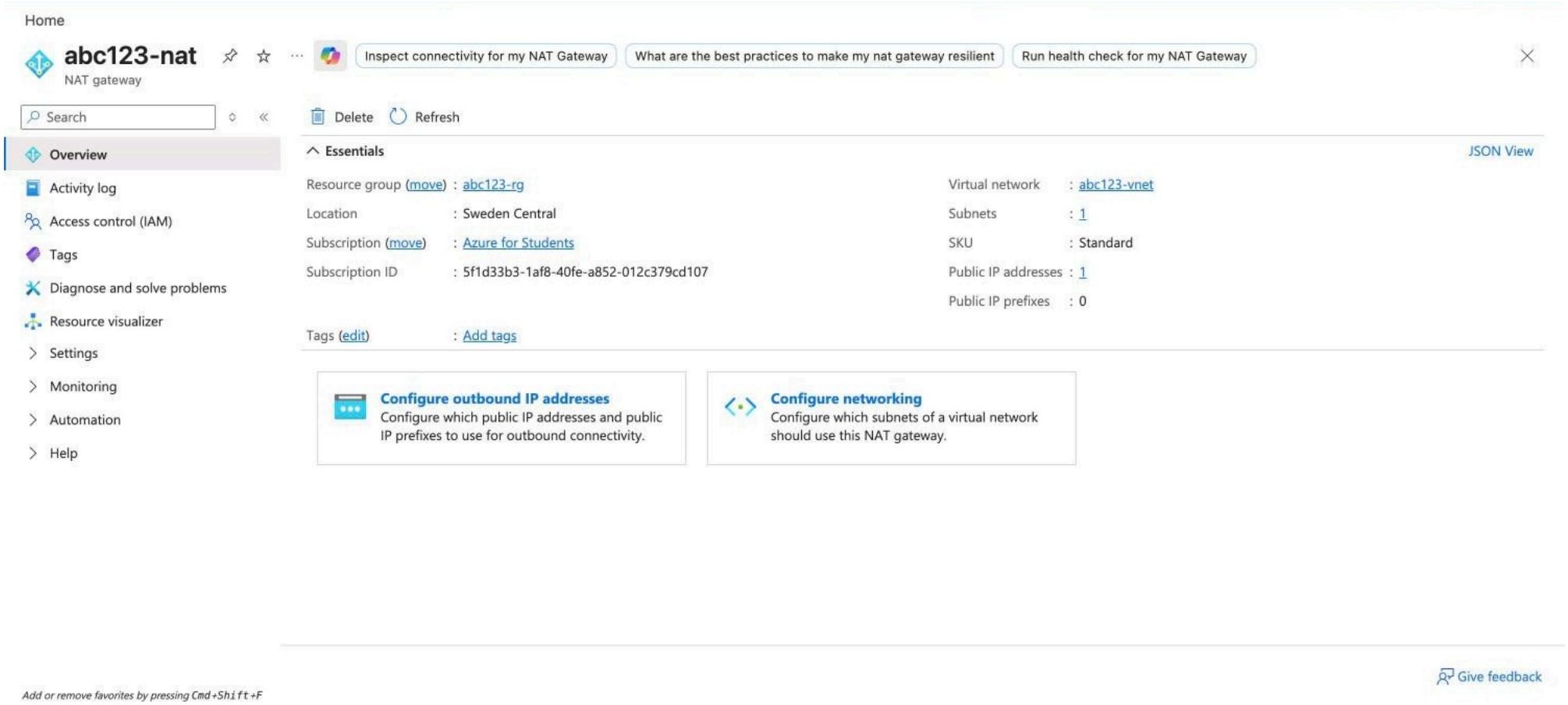
Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

Outputs:

lb_address = "135.225.95.152"
locked = false
```

- A second output, `locked`, so that the last line of every apply says which phase you are in
- Two of the three public IPs are now in use. `terraform state list | wc -l` says 20

# Step 4 — in the portal




*NAT gateways* → `abc123-nat` : *Subnets: 1* (the app subnet), *Public IP addresses: 1*. Nothing to `curl` yet; the way out matters only to machines, and there are none.

# Step 4 — what you wrote

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

- Every block in this file has that `count`. With `locked=false` each block makes one resource; with `locked=true` each makes none. The lock-down at the end of the lab is this variable and nothing else
- Four blocks: the gateway's `azurerm_public_ip.nat`, the `azurerm_nat_gateway.nat` itself, the association of the IP with the gateway, the association of the gateway with the **app subnet**. The db subnet is not associated: the database never needs a way out
- References to a counted resource carry the index: `azurerm_nat_gateway.nat[0].id`. Without `[0]` Terraform refuses the file

# Step 5 — the database: plan

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

```
  # azurerm_postgresql_flexible_server.db will be created
  # azurerm_private_dns_zone.db will be created
  # azurerm_private_dns_zone_virtual_network_link.db will be created

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

Azure Database for PostgreSQL, Flexible Server: you ask for a version and a size, Azure runs it, patches it, backs it up. On a private network it needs two helpers a VM would not: a **private DNS zone** in which the server's name resolves to its private address, and a **link** from that zone to your VNet.

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

# Step 5 — the database: apply

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

azurerm_private_dns_zone.db: Creation complete after 34s [id=…/privateDnsZones/privatelink.postgres.database.azure.com]
azurerm_private_dns_zone_virtual_network_link.db: Creation complete after 33s [id=…/virtualNetworkLinks/abc123-db-dns-link]
azurerm_postgresql_flexible_server.db: Still creating... [10s elapsed]
...
azurerm_postgresql_flexible_server.db: Still creating... [5m0s elapsed]
azurerm_postgresql_flexible_server.db: Creation complete after 5m5s [id=…/flexibleServers/abc123-db]

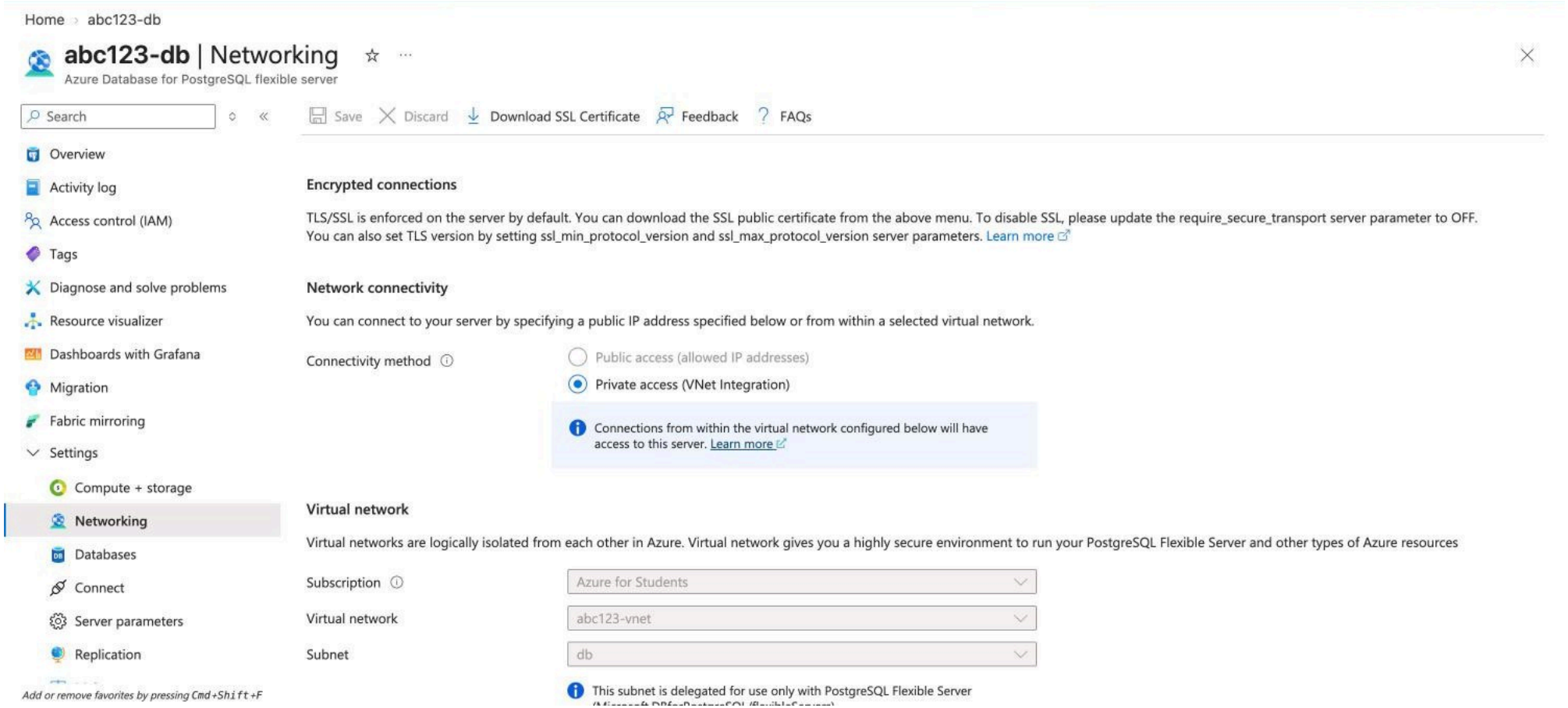
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

db_host = "abc123-db.postgres.database.azure.com"
lb_address = "135.225.95.152"
locked = false
```

`db_host` is the server's hostname. Nobody will type it anywhere: step 6 reads it from here. From your laptop it resolves to nothing; the zone is linked to the VNet, not to the internet. `terraform state list | wc -l` says 23.

# Step 5 — in the portal




*Azure Database for PostgreSQL flexible servers* → `abc123-db` → *Networking*: *Private access (VNet Integration)*, virtual network `abc123-vnet`, subnet `db`. No public endpoint exists.

# Step 5 — what you wrote

- `azurerm_postgresql_flexible_server.db` : `version = "16"` , `sku_name = "B_Standard_B1ms"` (the smallest burstable size, cents per hour), `storage_mb = 32768` , `administrator_password = var.db_password` , `public_network_access_enabled = false`
- `delegated_subnet_id = azurerm_subnet.db.id` and `private_dns_zone_id = azurerm_private_dns_zone.db.id` : two references, two ordering edges
- `depends_on = [azurerm_private_dns_zone_virtual_network_link.db]` : the server never references the link, yet fails to create without it. The one explicit dependency in the whole configuration; the comment above it says why
- `output "db_host" { value = azurerm_postgresql_flexible_server.db.fqdn }`

# Step 6 — the app VMs: plan

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

```
  # azurerm_linux_virtual_machine.app[0] will be created
  # azurerm_linux_virtual_machine.app[1] will be created
  # azurerm_network_interface.app[0] will be created
  # azurerm_network_interface.app[1] will be created
  # azurerm_network_interface_backend_address_pool_association.app[0] will be created
  # azurerm_network_interface_backend_address_pool_association.app[1] will be created

Plan: 6 to add, 0 to change, 0 to destroy.
```

Week 2's two VMs, with three differences: **no public IP**, **no SSH session**, and the beacon started by the VM itself at first boot. Three blocks with `count = 2`, six resources: `app[0]` and `app[1]` of each.

# Step 6 — the app VMs: apply

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

azurerm_network_interface.app[1]: Creation complete after 12s [id=…/networkInterfaces/abc123-app-2-nic]
azurerm_network_interface.app[0]: Creation complete after 23s [id=…/networkInterfaces/abc123-app-1-nic]
azurerm_network_interface_backend_address_pool_association.app[0]: Creation complete after 2s [id=…]
azurerm_network_interface_backend_address_pool_association.app[1]: Creation complete after 2s [id=…]
azurerm_linux_virtual_machine.app[1]: Creation complete after 49s [id=…/virtualMachines/abc123-app-2]
azurerm_linux_virtual_machine.app[0]: Creation complete after 49s [id=…/virtualMachines/abc123-app-1]

Apply complete! Resources: 6 added, 0 changed, 0 destroyed.
```

A minute and a half. Then about half a minute more while cloud-init installs Docker and pulls the image through the NAT gateway. `terraform state list | wc -l` says 29.

```
curl http://135.225.95.152/hello
```

The first tries time out like in step 3; then the beacon answers.

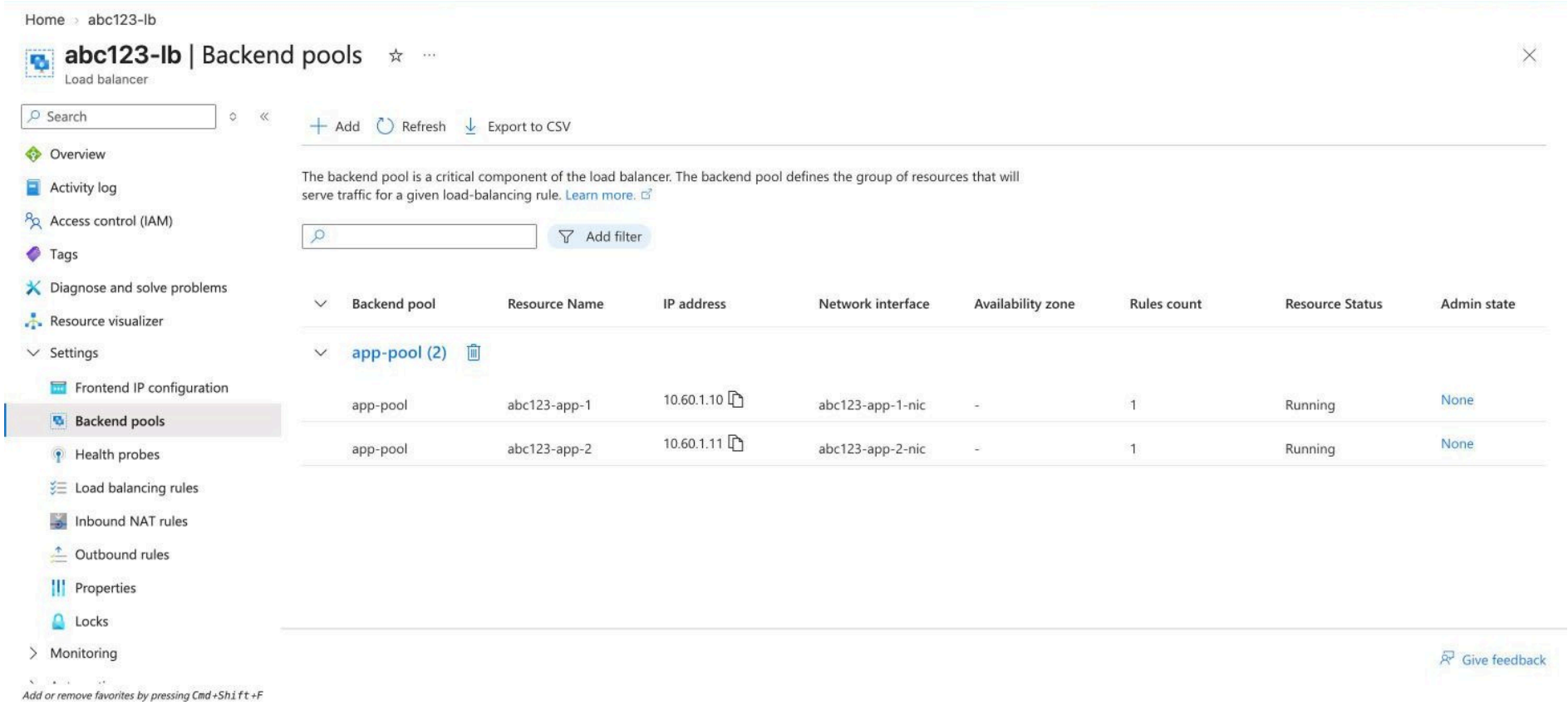
# Step 6 — what the beacon says

```
{ "beacon_id": "ac0ad2961f21", "hostname": "abc123-app-1",
  "interfaces": [ { "name": "eth0", "ips": ["10.60.1.10/24", ...] }, ... ],
  "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.postgres.database.azure.com:5432",
                "ok": true, "addr": "10.60.2.4" },
  "cloud": { "provider": "azure", "region": "SwedenCentral", "name": "abc123-app-1" } }
```

- `outbound` : the two TCP probes **succeed**. Expected: the NAT gateway is up, this is the build phase
- `db_check` : `ok: true` , `addr: 10.60.2.4` . The beacon resolved the database's name through the private zone to an address in the `db` subnet and connected. Nobody typed that name anywhere
- `interfaces` : `10.60.1.10` . No public one

Run it six times: `abc123-app-2` , `abc123-app-1` , `abc123-app-2` , … Two machines, one address.

# Step 6 — in the portal




`abc123-lb` → *Backend pools*: both VMs, `10.60.1.10` and `10.60.1.11`, *Running*. Open a VM: *Primary NIC public IP* is empty; *Networking* names the NAT gateway's address as its outbound one.

# Step 6 — what you wrote

- `azurerm_network_interface.app` : `count = 2` , `private_ip_address = cidrhost("10.60.1.0/24", 10 + count.index)` : `.10` and `.11` . A function and the index, not two copied blocks. No `public_ip_address_id`
- `locals { app_cloud_init = base64encode(templatefile("cloud-init-app.yaml", { ... })) }` : the boot file with its holes filled, `beacon_image` from your variable, `db_target` from `azurerm_postgresql_flexible_server.db.fqdn` . That reference is why the VMs are created **after** the server
- `azurerm_linux_virtual_machine.app` : `custom_data = local.app_cloud_init` , `admin_username = "student"` , your key, Ubuntu 24.04, `size = var.vm_size`
- `cloud-init-app.yaml` : install Docker, `docker run ... -e DB_TARGET=${db_target} ${beacon_image}` . What you typed over SSH last week, done by the VM at boot

# Step 7 — the bastion: plan

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

```
  # azurerm_linux_virtual_machine.bastion will be created
  # azurerm_network_interface.bastion will be created
  # azurerm_public_ip.bastion will be created

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

The app VMs never had a public IP, so the way in is one small VM in the `public` subnet, with the third and last public IP:

```
laptop ──22, from your IP only──▶ bastion 10.60.0.4 ──22, from 10.60.0.0/24 only──▶ app VMs
```

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

azurerm_public_ip.bastion: Creation complete after 3s [id=…/publicIPAddresses/abc123-bastion-ip]
azurerm_network_interface.bastion: Creation complete after 2s [id=…/networkInterfaces/abc123-bastion-nic]
azurerm_linux_virtual_machine.bastion: Creation complete after 50s [id=…/virtualMachines/abc123-bastion]

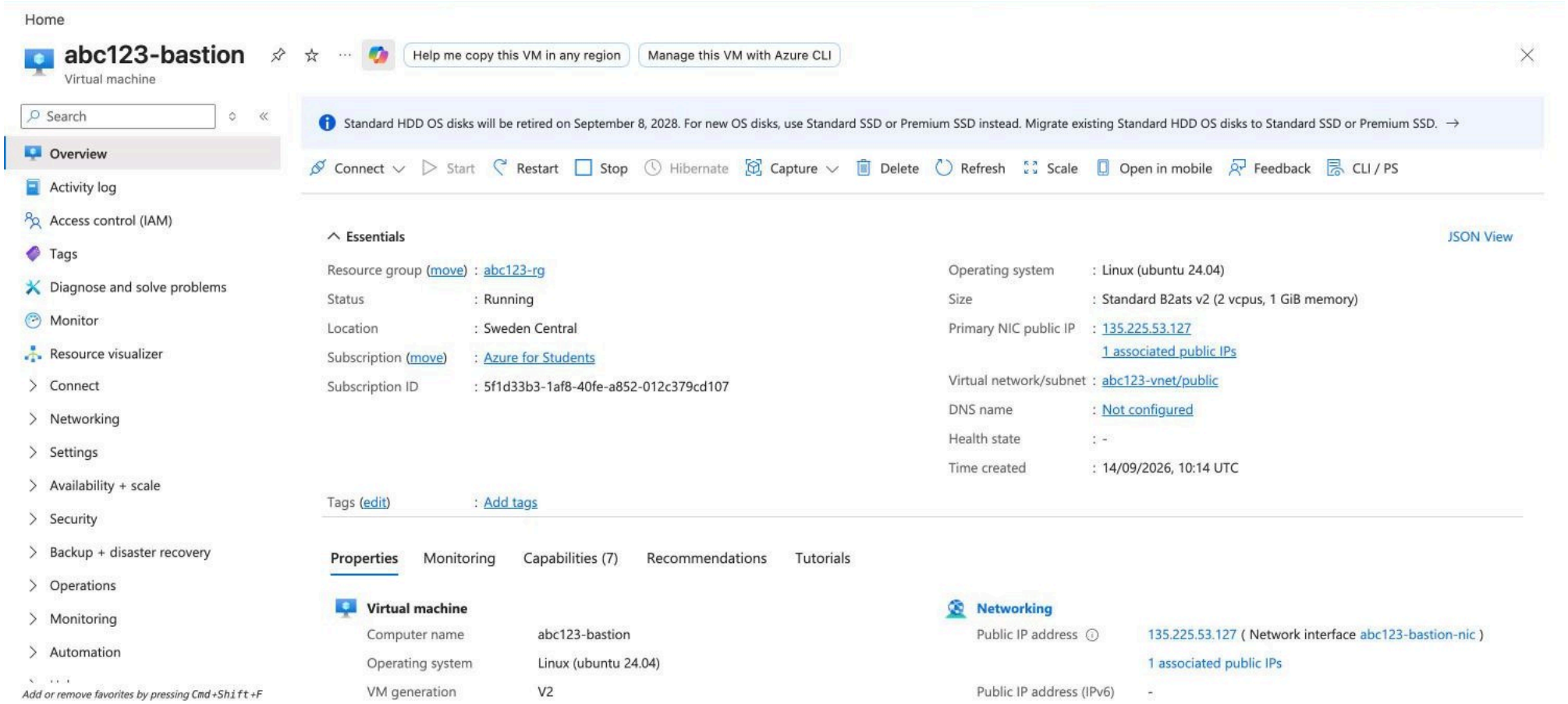
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

app_private_ips = [ "10.60.1.10", "10.60.1.11" ]
bastion_address = "135.225.53.127"
db_host = "abc123-db.postgres.database.azure.com"
lb_address = "135.225.95.152"
locked = false
ssh_command = "ssh -J student@135.225.53.127 student@10.60.1.10"
```

`terraform state list | wc -l` says **32**. 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 portal




`abc123-bastion` : *Primary NIC public IP* `135.225.53.127`, subnet `abc123-vnet/public`. The only VM in the subscription with an address of its own.

# Step 7 — look inside through the bastion

```
ssh -J student@135.225.53.127 student@10.60.1.10
```

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

```
student@abc123-app-1:~$ sudo docker ps
NAMES     IMAGE                                    STATUS
beacon    ghcr.io/siimrebane/iti8801-beacon:a3   Up 4 minutes
student@abc123-app-1:~$ nc -zv -w 5 abc123-db.postgres.database.azure.com 5432
Connection to abc123-db.postgres.database.azure.com (10.60.2.4) 5432 port [tcp/postgresql] succeeded!
student@abc123-app-1:~$ exit
```

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

```
ssh student@135.225.53.127
student@abc123-bastion:~$ nc -zv -w 5 abc123-db.postgres.database.azure.com 5432
nc: connect to abc123-db.postgres.database.azure.com (10.60.2.4) port 5432 (tcp) timed out
```

Same VNet, same database, different subnet: the `db` NSG's deny-VirtualNetwork rule. That failure is the database tier working as drawn.

# Step 7 — what you wrote

- `azurerm_public_ip.bastion`, and `azurerm_network_interface.bastion` with `public_ip_address_id`, at `10.60.0.4` in `azurerm_subnet.public`
- `azurerm_linux_virtual_machine.bastion`: same image, same size, same user and key as the app VMs, and no `custom_data`: nothing to install
- `output "bastion_address"`, and `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
```

```
  # azurerm_nat_gateway.nat[0] will be destroyed
  # (because index [0] is out of range for count)
  # azurerm_nat_gateway_public_ip_association.nat[0] will be destroyed
  # azurerm_public_ip.nat[0] will be destroyed
  # azurerm_subnet_nat_gateway_association.app[0] will be destroyed

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

Step 4's four resources, one variable. Nothing else changes: no VM is touched, the database is not touched. If the plan mentions `azurerm_linux_virtual_machine` or the database, stop and ask.

# Step 8 — lock down: apply

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

azurerm_subnet_nat_gateway_association.app[0]: Destruction complete after 4s
azurerm_nat_gateway_public_ip_association.nat[0]: Destruction complete after 11s
azurerm_nat_gateway.nat[0]: Destruction complete after 11s
azurerm_public_ip.nat[0]: Destruction complete after 11s

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

Outputs:
...
locked = true
```

Forty seconds. The app subnet has no route out any more. `terraform state list | wc -l` says 28.

# Step 8 — verify the lock-down

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

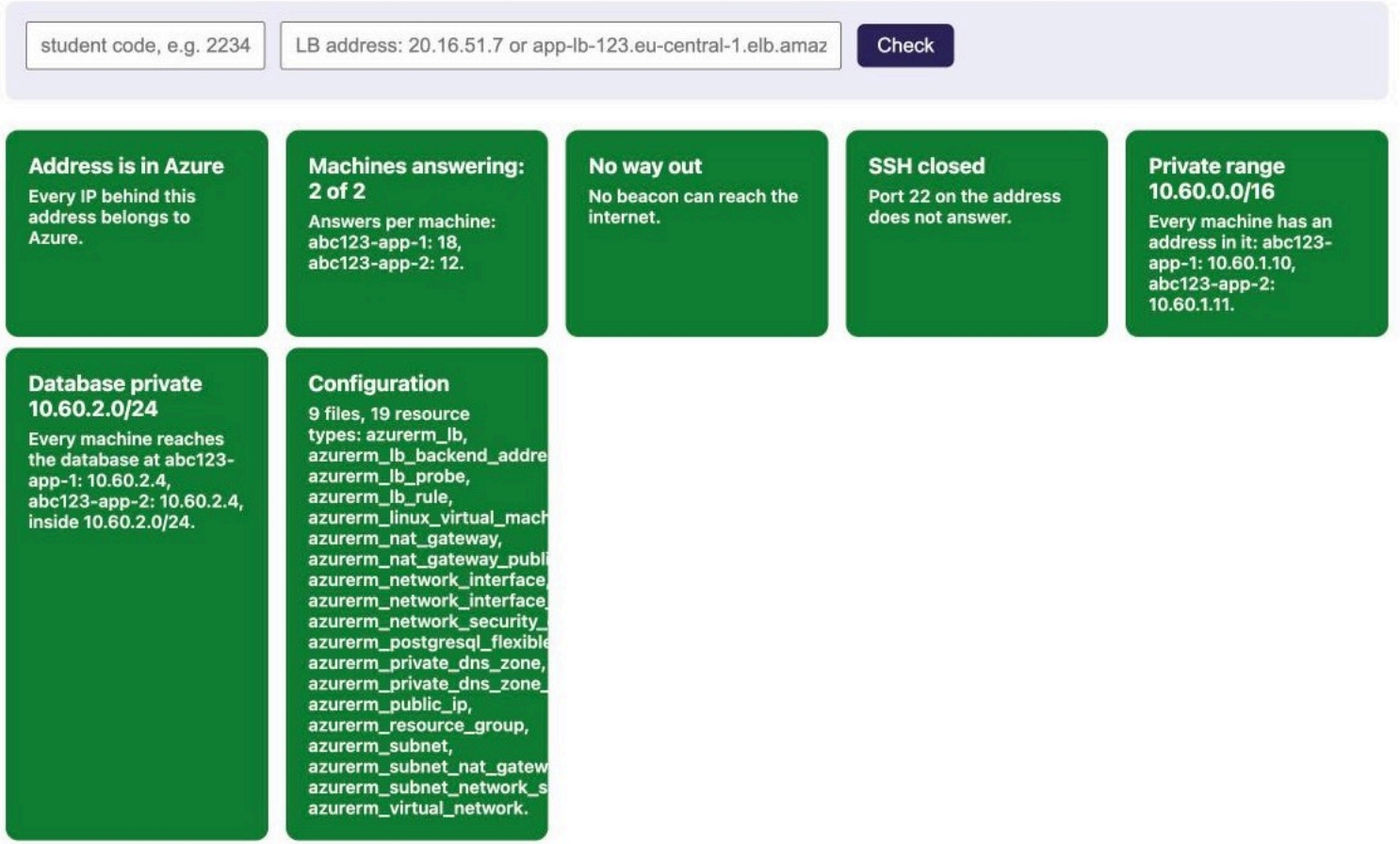
```
curl http://135.225.95.152/hello
```

```
  "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.postgres.database.azure.com:5432",
                "ok": true, "addr": "10.60.2.4" },
```

- Both TCP probes **fail** with a timeout. The app subnet has no way out
- `dns:example.com` still succeeds: Azure's resolver is a platform service inside the VNet, 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 VMs stays inside the VNet. The bastion still works: its subnet never used the NAT gateway

# Step 9 — the check

https://iti8801.digivader.com/lab3-validator/ · student code, the load balancer's address (`lb_address`, nothing else: no `http://`, no port), *Check*. 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/azure` except `me.tfvars` and the state: the nine `.tf` files, `cloud-init-app.yaml`, `.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: VMs, load balancer, database, NAT gateway, firewall, 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 VM has no healthy beacon yet; wait a minute, 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: no rule forwards 22 through the load balancer |
| Database private | `db_check` not ok: the app VMs booted before the database answered, or the beacon tag is not `a3` |
| Database private, address outside the range | the server is not in the delegated subnet; 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 portal: the next `plan` shows the drift.

# Step 9 — destroy

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

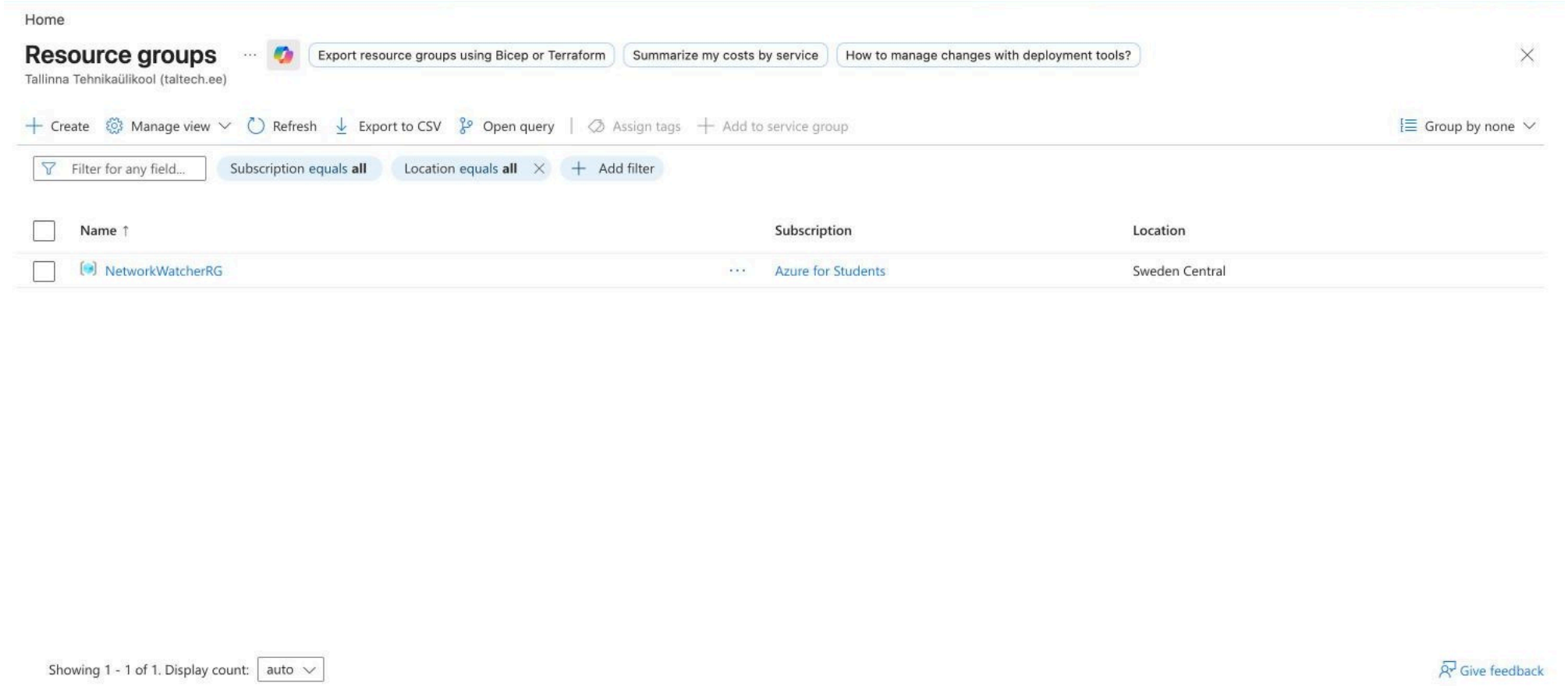
```
Plan: 0 to add, 0 to change, 28 to destroy.
  Enter a value: yes

azurerm_network_interface_backend_address_pool_association.app[0]: Destruction complete after 6s
...
azurerm_linux_virtual_machine.app[0]: Destruction complete after 40s
azurerm_postgresql_flexible_server.db: Destruction complete after 1m38s
azurerm_private_dns_zone_virtual_network_link.db: Destruction complete after 2m30s
azurerm_virtual_network.vnet: Destruction complete after 11s
azurerm_resource_group.rg: Destruction complete after 53s

Destroy complete! Resources: 28 destroyed.
```

Everything in the state, in reverse order, six and a half minutes. 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 portal




*Resource groups*: only `NetworkWatcherRG`, Azure's own. If your group is still there, something was created outside Terraform, and the portal 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
- `PublicIPCountLimitReached`: the quota. Something from an earlier week still exists; delete it in the portal, wait, `apply` again
- The database creation fails with a region or SKU message: ask; the answer depends on the region, and it goes into the file, not the portal
- The beacon never answers: the VMs booted before the database existed, or the image pull failed. `terraform apply -replace=azurerm_linux_virtual_machine.app[0] -var-file=me.tfvars` rebuilds one VM
- Truly stuck: `terraform destroy` and start over. Rebuilding is seven commands and about twelve 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 VM through the bastion, and saw the database refuse the bastion
- `locked=true` removed exactly four resources, 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 portal shows no resource group of yours



Questions