ITI8801 · Practice 2 (Azure)

Two VMs behind a load balancer, by hand

The Azure path. The AWS path is a separate deck; pick one cloud and follow it end to end.

Siim Rebane · TalTech · Autumn 2026

The task

Build this in the Azure portal, by hand:

Resource group	`iti8801-w2`, your week 1 region
Virtual network (VNet)	`iti8801-vnet`, address space `10.60.0.0/16`
Subnet	`app`, `10.60.1.0/24`, private
Virtual machines	`app-1`, `app-2`, B2ats_v2 or B1s, both in `app`, **no public IP**
Network security group (NSG)	`app-nsg` on the subnet: inbound TCP 80 from anywhere, nothing else
Software	the course beacon on both VMs, listening on port 80
Load balancer	Standard, one public IP, TCP 80 → both VMs, health probe HTTP `/hello`
Outbound	none: no public IPs, no NAT gateway, no outbound through the load balancer

Final state, and how it is proven

The load balancer's public IP is the only address that exists. Port 80 is the only port that answers. Every request lands on one of two machines that have no route to the internet.

The course validator sends requests to that address and checks five things:

- the address belongs to Azure
- exactly two distinct beacons answer behind it
- neither beacon can reach the internet
- port 22 on the address does not answer
- both beacons have an address in `10.60.0.0/16`

All five green: the check is stored on the server with a code, that code goes to Moodle, and you delete the resource group.

Why the build opens a door and closes it again

The final state has no way in and no way out. A machine like that cannot be built directly: installing the software needs both.

- **A way out**: `apt` and the Docker image pull need the internet. A VM with no public address sends the packet, and it is dropped at the edge of the VNet
- **A way in**: someone has to run the install. On this path that is you, over SSH

A public IP on each VM gives both at once and costs nothing. The production answers, a NAT gateway for the way out and Azure Bastion for the way in, bill by the hour and would exist only to be deleted an hour later.

So the build goes: public IPs and an SSH rule first, install, then remove both. The validator looks at the closed state only; having had a public IP during the build costs nothing on the check.

The plan on one slide

Phase	What	Why
1	Resource group, VNet, 1 subnet	the network
2	2 VMs **with temporary public IPs**	the machines; a way in and out for installing
3	1 NSG on the subnet, incl. a **temporary SSH rule**	the wall, tested against real machines
4	Install the beacon on both VMs over SSH	the software
5	Load balancer in front of the two VMs	the single entry point; watch it balance
6	**Lock down**: remove public IPs and the SSH rule	the assignment state
7	Validator check → **green**, copy its code	proof
8	**Delete the resource group**	it bills hourly

Region: the one you used in week 1. Each subscription offers its own list of regions; any of them works.

VM size: the free-tier B-series size that region offers, **B2ats_v2** or **B1s**.

Before you start: the public IP quota

Azure for Students allows **3 public IP addresses** per subscription. Today uses all three: one temporary IP per VM during the build, then one for the load balancer. With one address already in use, the load balancer's frontend IP in phase 5 is refused.

Public IP addresses in the portal must list **nothing** before you begin. Your A1 VM holds one.

Before you start: what may be deleted, and when

Anything submitted and **graded** may be deleted. Nothing needs to survive after grading: A1 is graded from the running VM, A2 from the validator's stored check.

- **A1 submitted and graded**: delete its resource group now, then start today
- **A1 submitted, not yet graded**: wait. A1 is graded within a day of submission, or on the spot in the lab
- **A1 not done yet**: do today first. Today ends with everything deleted, so the quota is free again for A1 afterwards

Deleting a resource group deletes every object in it, public IPs included. The quota is per subscription, so a public IP in any resource group counts.

Phase 1 — the resource group

Resource groups → Create: name `iti8801-w2`, region: **your week 1 region** → *Review + create* → *Create*.

A resource group is a folder. Every object you create in Azure (a VM, its disk, its network card, a public IP, a load balancer) must belong to exactly one. Everything today goes into this one, for two reasons:

- **Cost**: the cost page shows spending per resource group, so you see what today cost on one line
- **Deletion**: phase 8 deletes the group. That deletes every object inside it in one operation, and nothing is left behind to keep billing. Deleting a VM on its own leaves its disk, network card and public IP alive

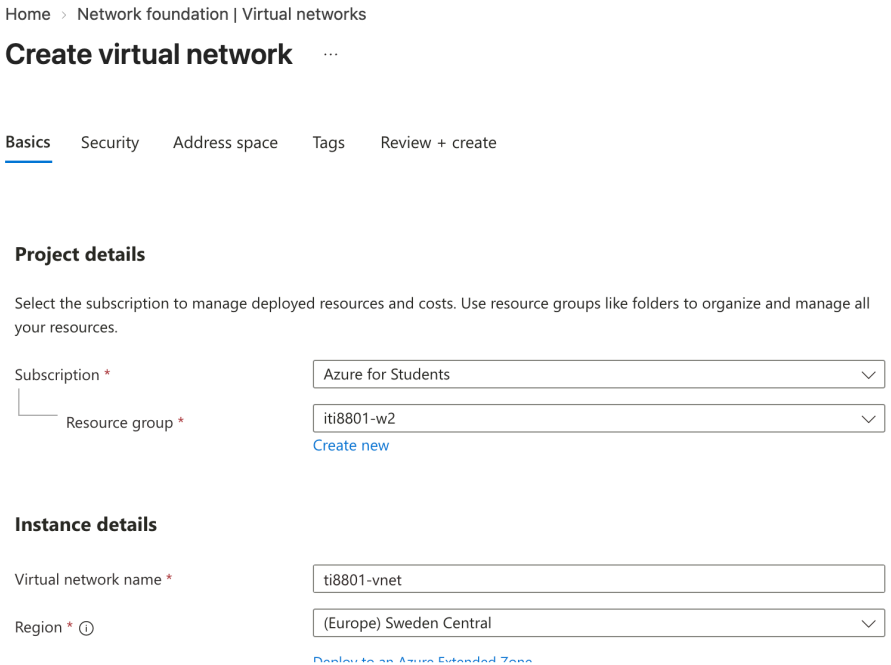
The group's region only says where its bookkeeping lives. But keep every resource in the same region anyway: a load balancer can only serve VMs in its own region.

Phase 1 — the virtual network: Basics

Virtual networks → Create. On the **Basics** tab:

- Resource group `iti8801-w2`, name `iti8801-vnet`
- Region: **the same as the resource group**. A VM can only join a network in its own region

Do not press *Review + create* yet. The two tabs that matter, *Security* and *Address space*, come first.



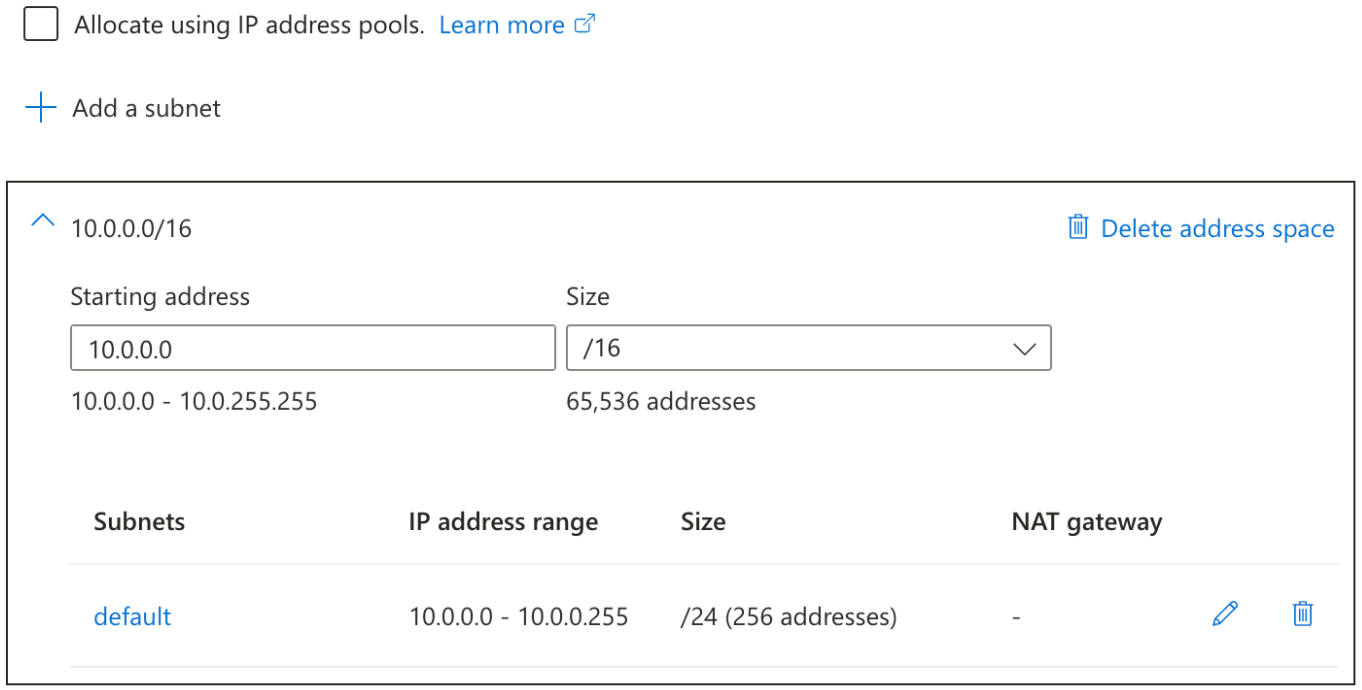
Phase 1 — the virtual network: Security

The **Security** tab offers four things. Leave all four **off**:

- **Virtual network encryption**: encrypts traffic between VMs inside the VNet. It requires *accelerated networking*, a feature the small free-tier VM sizes do not have, so it cannot be used today. Traffic to public IPs is not covered by it anyway
- **Azure Bastion**: the managed jump host from the lecture. We get in through temporary public IPs today and remove them; there is nothing for a bastion to do, and it bills by the hour
- **Azure Firewall**: a separate product, a firewall appliance with its own subnet and hourly price. It is not the Network Security Group; the NSG in phase 3 is free
- **DDoS Network Protection**: the attack-absorption service from the lecture, priced for companies

Phase 1 — the virtual network: Address space

The **Address space** tab is where the network's addresses are defined. As it opens: one address space `10.0.0.0/16`, entered as *Starting address* plus *Size*, with one subnet called `default` under it. Leave *Allocate using IP address pools* unchecked; it is a feature for organisations managing many networks.



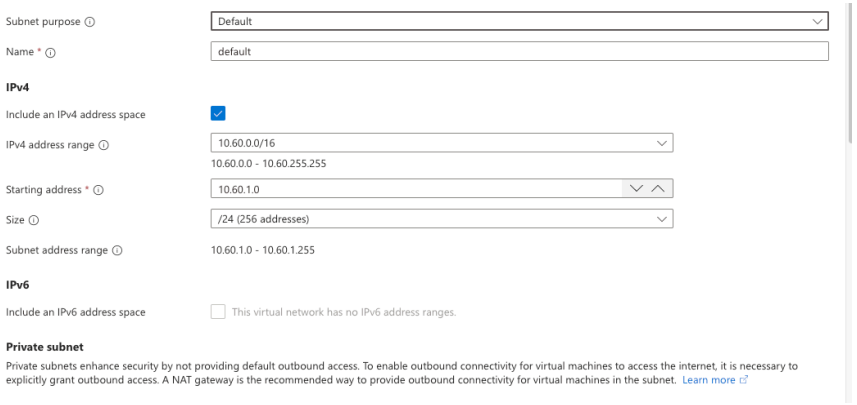
Phase 1 — the virtual network: the range

Replace what the portal pre-filled:

- *Starting address* `10.60.0.0`, *Size* `/16`. The pre-filled `10.0.0.0/16` would work just as well today. It is also the default everywhere, which makes it the range most likely to clash the day this network is connected to another one; so we pick a less common one, and we practise changing the field
- Delete the `default` subnet with the bin icon on its row

Then **+ Add a subnet**: the link is *above* the address space box, under the pools checkbox, not inside the box.

Phase 1 — the Add a subnet pane



- *Subnet purpose* **Default**. The other choices are templates for services that demand a subnet of their own with a fixed name (Azure Bastion, Azure Firewall, VPN gateway)
- *Name* `app`, *Starting address* `10.60.1.0`, *Size* `/24`
- **Private subnet**: ticked by default, leave it ticked. Off, Azure lends every VM a hidden shared public address for outbound traffic; on, the VM gets out only through a public IP, a NAT gateway or a load balancer. Phase 6 relies on that
- *Security* below it: NAT gateway, network security group, route table all stay **None**. The wall comes in phase 3

Phase 1 — the subnet

Subnet	Starting address	Size	Private	Who lives there
`app`	`10.60.1.0`	`/24`	yes	both VMs

One subnet, because everything that runs today has the same exposure: private, behind the load balancer. The load balancer itself is not in a subnet; on Azure it is a service that holds a public IP and forwards into the VNet. The lecture's database tier would be a second subnet with its own wall; it is not built today.

Then *Review + create* → *Create*.

Phase 1 — check yourself

Open the VNet. Its overview shows:

- Address space `10.60.0.0/16`
- *Subnets*: `app` `10.60.1.0/24`, **private**, with **no** security group and **no** route table. Nothing filters anything yet

Resource groups now also lists `NetworkWatcherRG`. Azure creates it by itself the first time a VNet appears in a region; it holds Network Watcher, the network diagnostics service. It costs nothing, holds no public IP, and comes back if deleted. Leave it.

Public IP addresses, portal-wide: still **empty**. If not, find and delete the leftover before phase 2; the quota is three, and the VMs need two of them now.

Phase 2 — why a public IP, and why only for now

Giving a server a public address is not what a production setup does. During today's build the VMs need two things, and a public IP is the only thing that provides both for free:

- **A way out**: `apt` and the Docker image pull need the internet. Without a public address a packet from a private VM has nothing to be translated to, and is dropped
- **A way in**: you, over SSH, to run the install

The production answers cost money by the hour and would exist only to be deleted an hour later: a **NAT gateway** for the way out, **Azure Bastion** for the way in.

The public IP on its own opens nothing, literally: a Standard public IP accepts no inbound traffic until a firewall rule allows it. It gives the machine an address the internet can route to; what answers at that address is decided by the firewall. In phase 6 the address is removed, and the machine vanishes from the internet.

Phase 2 — the two VMs

Virtual machines → Create → *Virtual machine*, twice:

	subnet	private IP	temporary
`app-1`	app	assigned by Azure, `10.60.1.4`	public IP
`app-2`	app	assigned by Azure, `10.60.1.5`	public IP

Azure keeps the first four addresses of every subnet for itself and hands out the next free one. Nothing today depends on which address a VM got.

Two VMs because a load balancer with one machine behind it balances nothing; with two you will see requests alternate between them in phase 5.

Phase 2 — the VM wizard: Basics

The same choices for both; only the name differs:

- Resource group `iti8801-w2`, name `app-1` / `app-2`, your region
- *Availability options*: **No infrastructure redundancy required**. The default pins the VM to one zone, and the free-tier size is the one most often out of capacity in a zone
- *Image*: **Ubuntu Server 24.04 LTS, x64 Gen2**, via *See all images* if it is not listed. Not *Ubuntu Pro*: Pro is a paid support subscription, billed per hour on top of the VM
- *Size*: the free-tier B-series size, **B2ats_v2** or **B1s**. *Azure Spot* stays off
- *Authentication*: **SSH public key**, the key from week 1 or a new pair the portal generates

Phase 2 — Basics, filled in

Instance details

Virtual machine name * : app-1

Region * : (Europe) Sweden Central
Deploy to an Azure Extended Zone

Availability options : No infrastructure redundancy required

Security type : Trusted launch virtual machines
Configure security features

Image * : Ubuntu Server 24.04 LTS - x64 Gen2
See all images | Configure VM generation

VM architecture * : Arm64 / x64

Size * : Standard_B2ats_v2 - 2 vcpus, 1 GiB memory (US$7.10/month)
See all sizes | Customize cores

Run with Azure Spot discount :

Administrator account

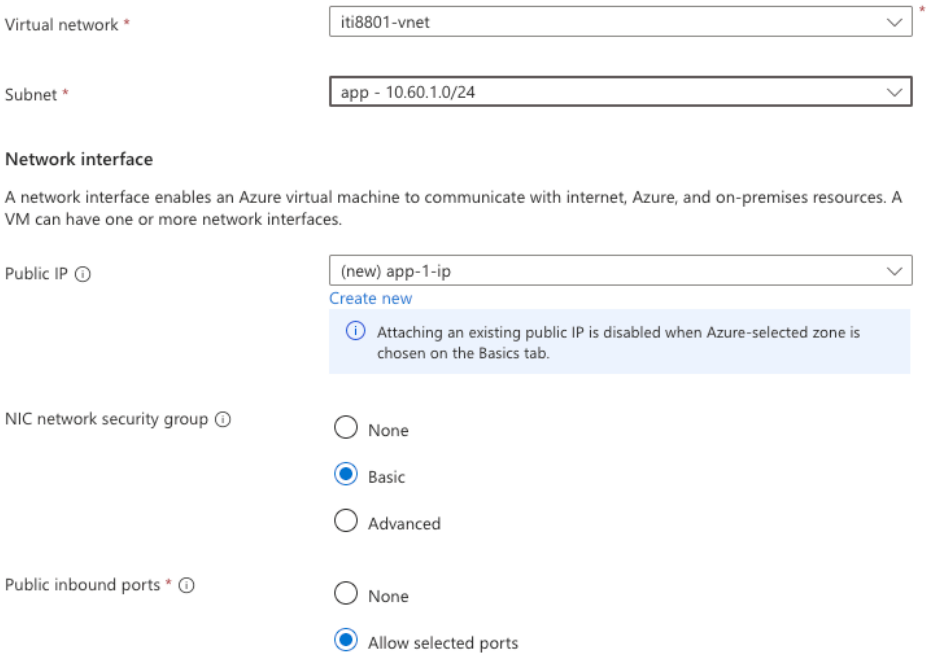
Authentication type * : SSH public key

Phase 2 — the VM wizard: Networking

- *Virtual network* `iti8801-vnet`; *Subnet* `app`
- *Public IP*: **(new)**, the default. Temporary; deleted in phase 6
- *NIC network security group*: **None**. The default, *Basic*, creates a second firewall on the network card with SSH open to the whole internet; the yellow warning is about that rule, and it disappears with *None*. The wall goes on the subnet in phase 3; a second NSG on the card would stack on top of it
- *Load balancing*: **None**. This only puts the VM behind a load balancer that already exists, and ours does not yet. The two choices offered are the lecture's two kinds: *Azure load balancer* forwards TCP connections (layer 4), *Application gateway* reads HTTP (layer 7). Phase 5 creates the first kind and adds the VMs to it there
- *Delete public IP and NIC when VM is deleted*, *accelerated networking*: leave unticked. *Disks*, *Management*, *Monitoring*, *Advanced*: defaults

Phase 2 — Networking, as it opens

The default is the wrong choice. Select **None**:



Then *Review + create* → *Create*, and *Download private key* if the portal generated one.

Phase 2 — check yourself

- Both VMs show a private IP from the `app` subnet and a public IP
- `ssh azureuser@<public-ip>` **hangs and times out**, for both. That is correct

The public IPs the portal creates are *Standard* SKU, and a Standard public IP is **closed to all inbound traffic until a Network Security Group rule allows it**. With no NSG anywhere, nothing gets in: not SSH, not port 80, nothing. The address exists and is routable; the door in front of it is shut. Phase 3 opens exactly the two ports we need.

If SSH instead answers *Permission denied (publickey)*, the address is reachable and the problem is the key: the user is `azureuser` unless you changed it, and a downloaded `.pem` needs `chmod 600` and `ssh -i <file>`.

Write down both private IPs and both public IPs. They're in the steps from here on.

Phase 3 — the firewall

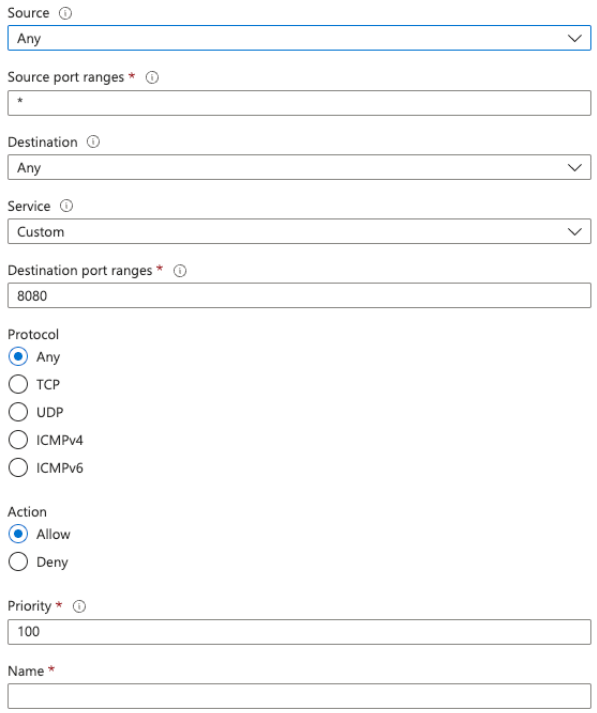
Network security groups → Create: the form asks only for a name, `app-nsg`, the resource group and the region. The rules come after creation: open the NSG → *Settings* → *Inbound security rules* → *Add*, twice:

Priority	Rule	Name
100	allow TCP **80** from anywhere	`http-in`
110	allow TCP **22** from **My IP address**	`ssh-temp`

Rule 100 is the one opening this setup keeps: the load balancer delivers requests on port 80. Rule 110 is temporary; it is deleted in phase 6. Everything else stays shut: the built-in *DenyAllInBound* rule at the bottom of the list, and the Standard public IP's own default of accepting nothing an NSG has not allowed.

Phase 3 — the Add rule form

Two port fields. *Source port* is the client's side, picked at random by the client for every connection: always `*`. *Destination port* is your VM's side, where the service listens: the one you filter. The defaults (8080, Any) are placeholders.



Fill, for `http-in`:

- *Source* **Any**, *Source port ranges* `*`, *Destination* **Any**
- *Service* **HTTP**: fills port 80 and TCP
- *Action* Allow, *Priority* **100**, *Name* `http-in`

For `ssh-temp`, the same except: *Source* **My IP address**, *Service* **SSH** (22, TCP), *Priority* **110**, *Name* `ssh-temp`.

Phase 3 — the NSG as created

Before your rules, the overview shows six rules that were there from the start:

Filter by name | Port == all | Protocol == all | Source == all | Destination == all | Action == all

Priority	Name	Port	Protocol	Source	Destination	Action
Inbound Security Rules						
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancer...	Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny
Outbound Security Rules						
65000	AllowVnetOutBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowInternetOutBound	Any	Any	Any	Internet	Allow
65500	DenyAllOutBound	Any	Any	Any	Any	Deny

- **Inbound**: anything from inside the VNet allowed, the platform's load balancer probes allowed, everything else denied. Your 100 and 110 sit above these and win
- **Outbound**: the internet is *allowed*. The NSG lets packets leave; whether they can is decided by the address, which is why removing the public IP in phase 6 cuts the VMs off with no rule change
- *Essentials* at the top says how many rules are yours and which subnets the NSG is attached to. After phase 3 both must be non-zero

Phase 3 — attach the wall to the subnet

An NSG with rules in it does nothing on its own. It filters traffic only where it is attached, and until then the VMs are exactly as closed as before.

`app-nsg` → *Settings* → *Subnets* → *Associate* → virtual network `iti8801-vnet`, subnet `app` → OK.

- Attached to the **subnet**, so it covers both VMs now and any VM added to that subnet later
- Not to the network cards: that is the other place an NSG can go, and one NSG per VM would mean the same rules kept in two places
- The *Essentials* line on the NSG overview now reads *1 subnet*, and SSH to both VMs starts answering within a minute or so

Phase 3 — check yourself

The wall is up around real machines, so test it:

- `ssh azureuser@<public-ip>` **now works** for both VMs: rule 110 from your address is the first thing that let anything in
- From your phone's network, or any other address, port 22 does not answer: the built-in deny
- The `app` subnet lists `app-nsg` under *associations*; the VMs' network cards list **no** NSG
- No warning triangle on `ssh-temp` in the rules list. Azure marks a rule that opens 22 to *Any* with one; if it is there, the source is not your address

Common mistake: the rule's source. *My IP address* is your address right now; on a different network tomorrow the rule no longer matches you, and SSH stops working. That is the rule working, not breaking.

Phase 4 — install the beacon

The same commands on `app-1` and then `app-2`:

```
ssh azureuser@<public-ip>
sudo apt update && sudo apt install -y docker.io
sudo docker run -d --name beacon --restart=always --network host ghcr.io/siimrebane/iti8801-beacon:latest
```

`--network host` puts the container on the VM's own network instead of Docker's private one, so the beacon sees the VM's real interface and listens on the VM's port 80 directly. `curl localhost/hello` on each VM returns JSON with a `beacon_id`. The two IDs differ: the beacon makes one up at every start, and that is how anything in front can tell the machines apart.

Phase 4 — read what the beacon says

The beacon's answer is the machine's view of itself from inside. Ask for it on the VM, or from your laptop through the load balancer once it exists:

```
curl localhost/hello                  # on the VM
curl http://<LB-PUBLIC-IP>/hello      # from outside, through the LB
```

Right now it should show:

- `interfaces`: `eth0` with the private `10.60.1.4` (or `.5`) and a `fe80::` address, IPv6's link-local kind, meaningful only on this subnet; and `docker0`, Docker's own bridge. **Not** the public IP: the platform translates it, the VM never sees it
- `hostname`: the VM's name, since the container shares the VM's network
- `outbound`: the two TCP targets **succeed**, expected; the temporary public IP gives the VM a way out. `dns:example.com` succeeds too, and keeps succeeding after the lock-down: Azure's resolver `168.63.129.16` is a platform service inside every VNet, not the internet, and the validator ignores it

Phase 5 — what a health probe is

A load balancer has a list of backends and no idea which of them work. So it asks, all the time: every few seconds it sends each backend one request of its own and looks at the answer.

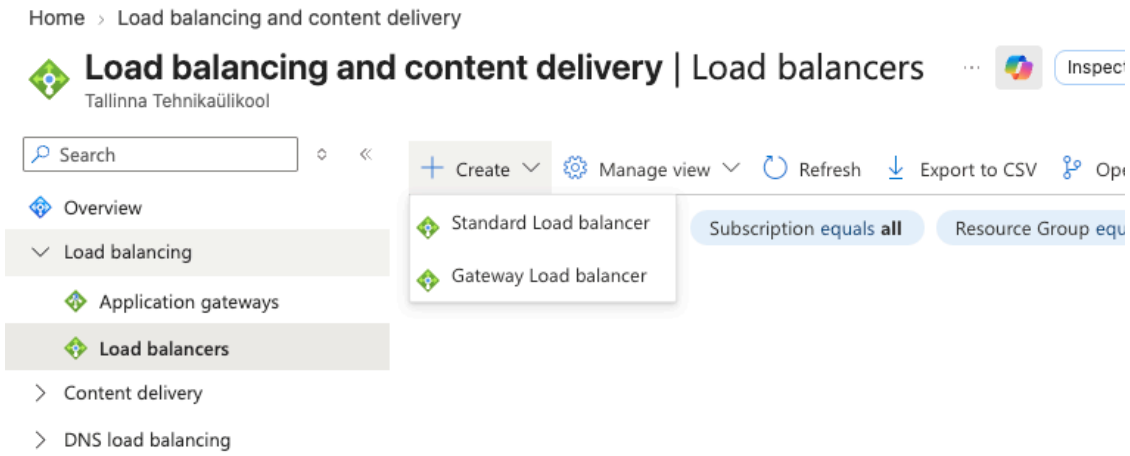
- **TCP probe**: opens a connection to the port. Proves that something listens, nothing more. A crashed application whose port is still held open passes
- **HTTP probe**: sends `GET <path>` and requires status **200**. Proves the application itself answers. Today's path is `/hello`, the beacon's own endpoint

A backend that fails **two probes in a row** is marked unhealthy: it gets no new connections until it answers again. Nothing else changes; the VM keeps running, and the probe keeps asking. With the interval at 5 seconds, a dead backend is out of rotation in about 15 seconds and back in as soon as it answers.

The probes come from Azure's address `168.63.129.16`. That is why the NSG's built-in rule *AllowAzureLoadBalancerInBound* exists: without it the probes would be denied, every backend would be unhealthy, and the load balancer would forward nothing.

Phase 5 — the load balancer

Load balancers → *Create* → **Standard Load balancer** (not Gateway):

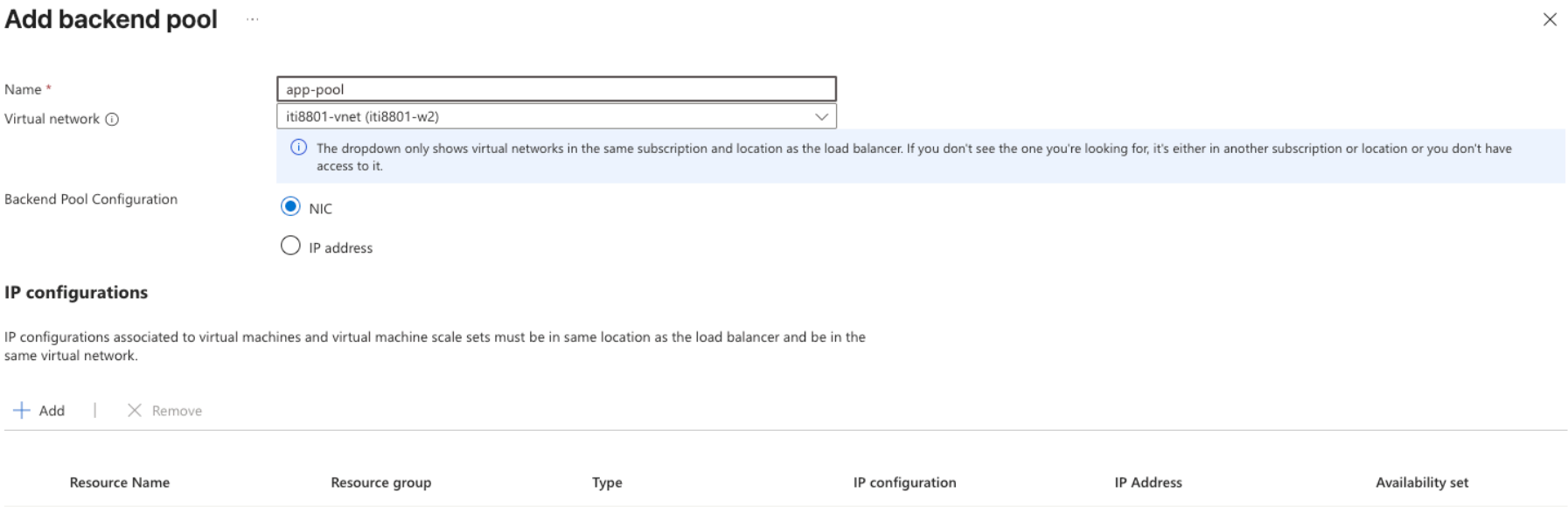


One wizard with tabs: *Basics*, *Frontend IP configuration*, *Backend pools*, *Inbound rules*, *Outbound rules*, *Review + create*. Each of the first four needs something from you.

Phase 5 — the first three tabs

- *Basics*: resource group `iti8801-w2`, name `app-lb`, your region, *SKU* **Standard** (SKU is Azure's word for the product tier), *Type* **Public** (the default is *Internal*, which gets no public address), *Tier* Regional
- *Frontend IP configuration* → *Add*: name `app-lb-frontend`, IPv4, IP address. Under *Public IP address* → *Create new*: name `app-lb-ip`, the rest at its defaults (Standard, Static, Zone-redundant) → *Save*, *Add*. This is the third and last of the quota
- *Backend pools* → *Add*: name `app-pool`, virtual network `iti8801-vnet`, configuration **NIC** (network interface card: the VM's network card). Under *IP configurations* → *Add*: tick `app-1` and `app-2`, *Add*, then *Save*

Phase 5 — the backend pool, filled in



Under *IP configurations* → *Add*, both VMs appear because both have a network card in `iti8801-vnet`. After *Add* the table lists `app-1` and `app-2` with their private addresses.

Phase 5 — the rule and the probe

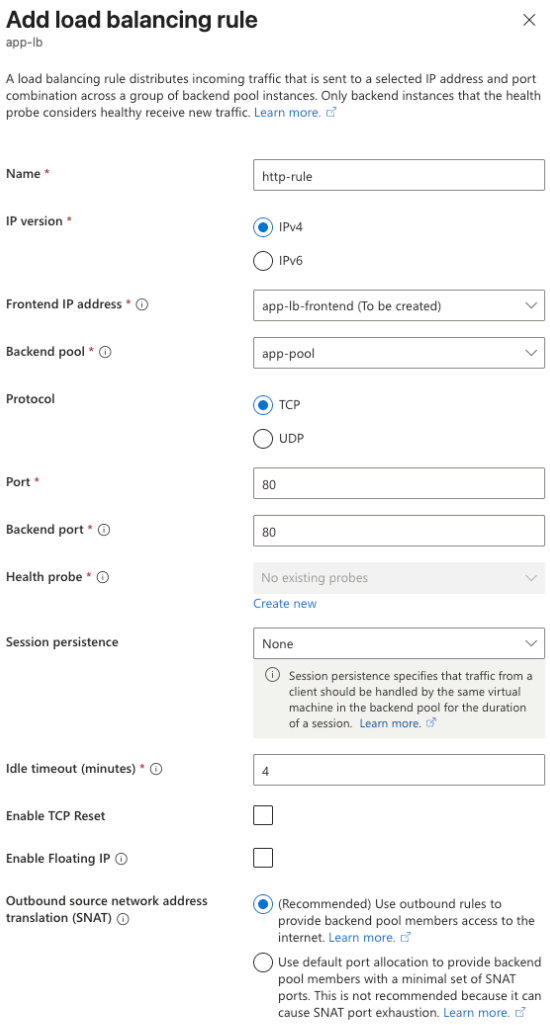
- *Inbound rules* → *Add a load balancing rule*: name `http-rule`, IPv4, frontend `app-lb-frontend`, backend pool `app-pool`, TCP, port **80**, backend port **80**
- *Health probe* → *Create new*: name `hello-probe`, protocol **HTTP** (switch it from TCP; the *Path* field appears only then), port 80, path `/hello`, interval 5 → *Save*
- Session persistence None, idle timeout 4, TCP reset and floating IP off: the defaults
- *Outbound source network address translation (SNAT)*: leave **(Recommended) Use outbound rules**. The other option gives the VMs internet access through the LB, and the validator will see it
- *Save*. Nothing on the *Outbound rules* tab, then *Review + create* → *Create*

Wait for both backends to turn healthy, then from **your laptop**:

```
curl http://<LB-PUBLIC-IP>/hello
```

The same beacon JSON, now through the load balancer.

Phase 5 — the rule, filled in



Everything on this form except the name, the two selections and the two ports is a default.

The health probe is still empty here: it is created from the *Create new* link, and the form refuses to save without one.

The SNAT choice at the bottom is the default. Leave it.

Phase 5 — watch it balance

Ask the load balancer ten times and look only at which beacon answered:

```
for i in $(seq 10); do curl -s http://<LB-PUBLIC-IP>/hello | grep -o '"beacon_id":"[^"]*"'; done
```

Two different IDs, in no fixed order. Azure spreads connections by a hash of source and destination addresses and ports; each new connection from your laptop lands on one of the two. Ten in a row on the same machine does happen: consecutive connections from one client use neighbouring source ports, and the hash can map a run of them to one backend. Run the loop again.

Now the health check. On `app-1`: `sudo docker stop beacon`. Within about 15 seconds (two failed probes) the loop shows only `app-2`'s ID; the LB stopped sending to a backend that does not answer. `sudo docker start beacon` on `app-1`, and both IDs are back. This is the mechanism that keeps a service up while one machine is down.

Phase 6 — remove the temporary access

In this order, for `app-1` and then `app-2`:

1. *Virtual machines* → `app-1` → *Overview*. In *Essentials*, *Primary NIC public IP* shows the address as a link. Click the address: this opens the public IP as its own resource, on its *Configuration* page
2. In the public IP's left menu click **Overview**. Its toolbar has the buttons: **Dissociate** → *Yes*. The VM keeps running and loses its address
3. Same toolbar: **Delete** → *Yes*. The quota slot is free again
4. Back on the VM's *Overview*, *Primary NIC public IP* now shows `–`

Phase 6 — the SSH rule, and the proof

The firewall rule: *Network security groups* → `app-nsg` → *Inbound security rules* → tick `ssh-temp` (priority 110) → *Delete*. The validator cannot see this rule: it probes port 22 on the load balancer's address, and no rule forwards that port. Delete it anyway; a rule that allows SSH from one laptop is an entry point waiting for a public IP to be added back.

Public IP addresses, portal-wide, now lists exactly one: `app-lb-ip`.

Then verify the lock from your laptop:

```
curl http://<LB-PUBLIC-IP>/hello       ← still answers (through the LB)
ssh azureuser@<old-public-ip>          ← dead (no address)
```

Within a minute both beacons' TCP `outbound` results change to **failing**; the beacon re-tests every 60 seconds. Only `dns:example.com` stays OK, the platform resolver. This is the intended state; it is what A2 requires.

Phase 7 — the check

Course validator: **https://iti8801.digivader.com/lab2-validator/**

1. Enter your student code + `<LB-PUBLIC-IP>` (the address only, no `http://`)
2. The server checks, through your LB:

Check	What it proves
LB address belongs to Azure	it is a real cloud load balancer
Many requests reach **exactly two** beacons	the LB leads to your two VMs, and both are healthy
Beacons: no outbound anywhere	the VMs have no way out
Port 22 on the LB address does not answer	no SSH through the entry point
Every beacon has an address in `10.60.0.0/16`	the VMs are in the VNet this deck builds

3. Five green boxes → **passed**. Under them is the **check code**, a long random id: copy it, that code is your submission

Phase 7 — not green?

Each box maps to one phase of today: the address to phase 5, the two machines to phases 2 to 5, the way out to phase 6, the private range to phases 1 and 2. Fix, press *Check again*.

Every check is stored with its own code and nothing is ever overwritten, so check as often as you like and submit the code of a green one.

Phase 7 — passed

ITI8801 · Assignment 2 check

Enter your student code and the **public address of your load balancer**: an IP address or a DNS name, nothing else (no `http://`, no port, no path).
Every check is recorded with its own code; nothing is overwritten, and you can check as often as you like.

asdfas | 135.116.3.35 | Check

Address is in Azure
Every IP behind this address belongs to Azure.

Machines answering: 2 of 2
Answers per machine: app-1: 12, app-2: 18.

No way out
No beacon can reach the internet.

SSH closed
Port 22 on the address does not answer.

Passed. This check is recorded: student code ASDFAS, address 135.116.3.35, 2026-09-05T23:50:55Z. Its code:

`a994013a-248c-460e-b276-6461a39c58c7`

To submit this as your Assignment 2 result, paste this code into Moodle. Nothing else is needed, and you can delete your infrastructure now.

Check again

Phase 8 — delete everything

Everything you built bills by the hour, and your green check is already saved.

- *Resource groups* → `iti8801-w2` → **Delete resource group**. One operation, everything in it dies
- *Public IP addresses* afterwards: empty again
- Check *Cost Management* tomorrow: the line should be flat again. If the budget alarm emails you later, something from today is still running



The group name must be typed in full; *Delete* stays grey until it matches.

Summary — what you built

By hand, in ~90 minutes:

resource group → VNet → subnet → 2 VMs → NSG rules → beacon on both → LB over the two → lock-down

Keep today's notes. Every step you did by hand created one named object, and that list of objects is the whole setup.

Done when

- A **green check** on the validator (stored on the server; a screenshot is optional)
- **Everything deleted**, cost page flat
- A2 submission in Moodle: **the check code** of a green check, nothing else

Deadline: **Sunday of week 3, 23:59.**