THOMAS MORE

Lab 2.3

Segmenting the Internal Backend

Toepasbare Cloud Security voor IT-Professionals

2025 - 2026

Contents

1. Segmenting the Internal Backend

1.1. Context

In Lab 2.2 you proved that the internal backend is directly reachable from the public internet. You also saw that logging the event does not block it.

In this lab you will add a network-level restriction so that only the public backend API can reach the internal backend. The direct path from the public internet should be blocked. The expected path through the public backend should continue to work.

1.2. Learning goals

By the end of this lab, you should be able to explain:

- How to configure Regional VNet Integration for an App Service.
- Why `WEBSITE_VNET_ROUTE_ALL=1` is required for access restrictions to work.
- How an App Service access restriction limits inbound traffic.
- How VNet integration allows one App Service to call another through the private network.
- Why blocking one path does not break the application.
- The difference between an access restriction and an NSG rule.
- How to verify that a fix works without breaking the expected path.

1.3. Estimated time

75 minutes

Suggested timing:

Time	Activity
10 minutes	Add VNet integration to public backend
15 minutes	Add access restriction to internal backend
20 minutes	Test and verify after the fix
20 minutes	Reflection questions
10 minutes	Group discussion

1.4. Before you start

Make sure you have:

- [ ] The `public_backend_url` and `internal_backend_url` from Lab 2.1
- [ ] The VNet and app integration subnet name from Lab 2.1 (or from the Portal: VNet → Subnets)
- [ ] Azure Portal access

> **Note**
>
> The goal of this lab is not to block everything. The goal is to block the unexpected path while keeping the expected path working.

2. Micro-theory: VNet integration and App Service access restrictions

2.1. VNet integration

App Services are PaaS offerings that run outside your VNet by default. When you enable Regional VNet Integration, the App Service gets a network connection into a designated subnet (the “app integration subnet”).

This allows the App Service to make outbound calls to resources inside the VNet, such as:
- Virtual machines
- Private endpoints (for DB storage, Key Vault, etc.)
- Other App Services with access restrictions that allow the VNet subnet

2.2. Route-all setting

By default, VNet integration only routes RFC 1918 private IP ranges through the VNet:
- 10.0.0.0/8
- 172.16.0.0/12
- 192.168.0.0/16

Calls to public IP addresses (including Azure public endpoints like `*.azurewebsites.net`) still go directly from the App Service, bypassing the VNet integration.

The app setting `WEBSITE_VNET_ROUTE_ALL=1` changes this behavior:
- All outbound traffic (private and public) is routed through the VNet integration subnet
- The App Service appears to be calling FROM the VNet subnet
- This allows access restrictions on the destination App Service to recognize the caller as coming from the allowed subnet

> **Note**
>
> Critical: Without `WEBSITE_VNET_ROUTE_ALL=1`, the VNet subnet access restriction will NOT work for App Service-to-App Service communication via public endpoints. The calling App Service must have this setting enabled.

2.3. Access restrictions

App Service access restrictions let you control which sources can reach your app’s public endpoint. When you allow a VNet subnet as the source:
- Traffic FROM that subnet is allowed
- This includes App Services with Regional VNet Integration to that subnet (if they have `WEBSITE_VNET_ROUTE_ALL=1`)
- All other traffic is blocked (when default action is Deny)

The combination of:
1. Public backend has VNet integration to `snet-app-integration`
2. Public backend has `WEBSITE_VNET_ROUTE_ALL=1`
3. Internal backend allows `snet-app-integration` in access restrictions

…results in: public backend can call internal backend, but direct internet calls are blocked.

3. Confirm your Azure context

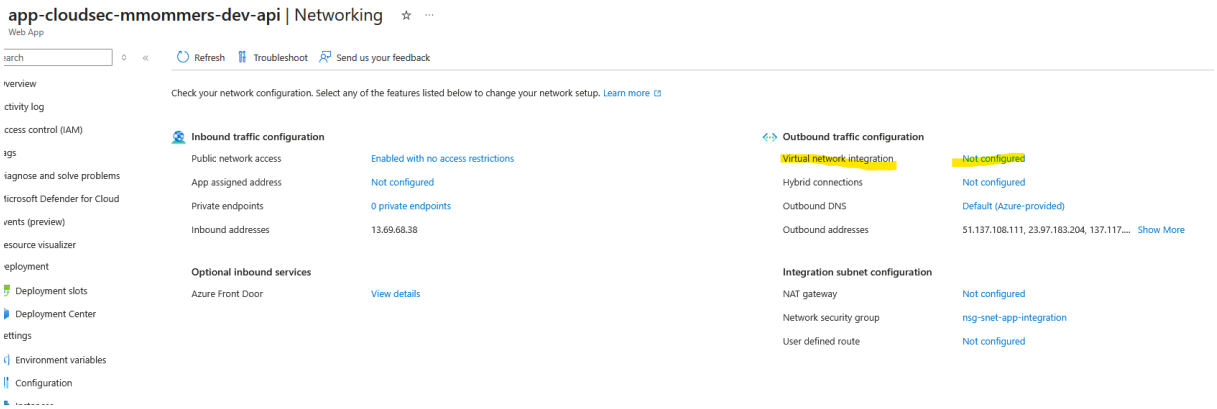
```
az account show --output json
```

Item	Value
Subscription name	
Resource group name	

4. Part 1 — Add VNet integration to the public backend

Before adding the access restriction on the internal backend, the public backend needs to be able to call resources inside the VNet. We configure VNet integration to enable this.

4.1. Add VNet integration

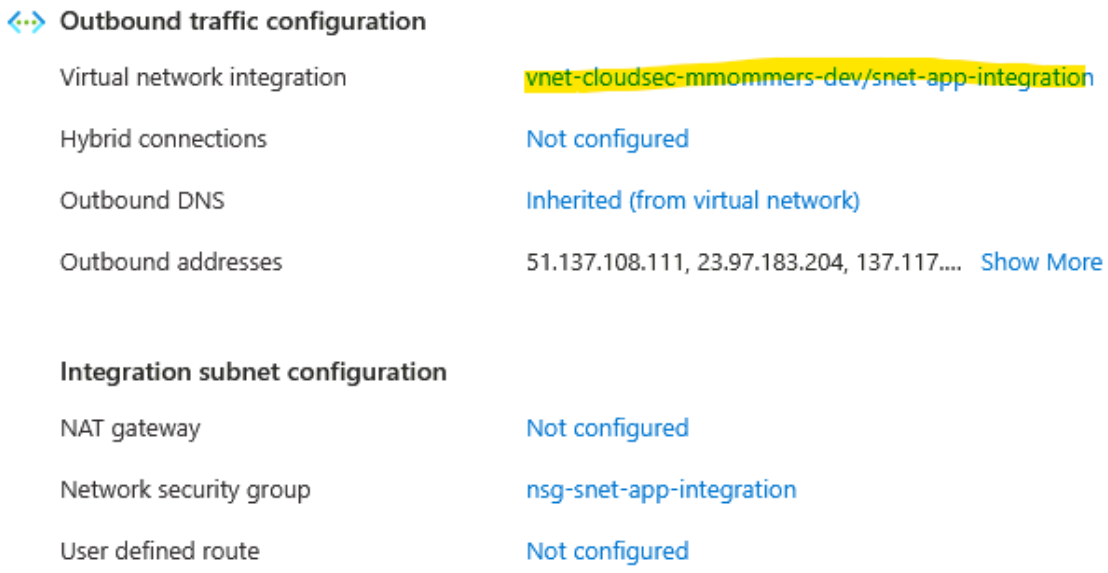


1. Navigate to the public backend App Service in the Azure Portal
2. Go to **Settings → Networking**
3. Click on the **VNet integration** section for Outbound traffic (not the Private Endpoint section for Inbound traffic)
4. Click **+ Not Configured**
5. Configure the integration:

Setting	Value
Virtual Network	Select your Day 2 VNet (`vnet-cloudsec-<prefix>-dev`)
Subnet	`snet-app-integration`

6. Click **OK**

Wait for the integration to be created (usually under 30 seconds)“.

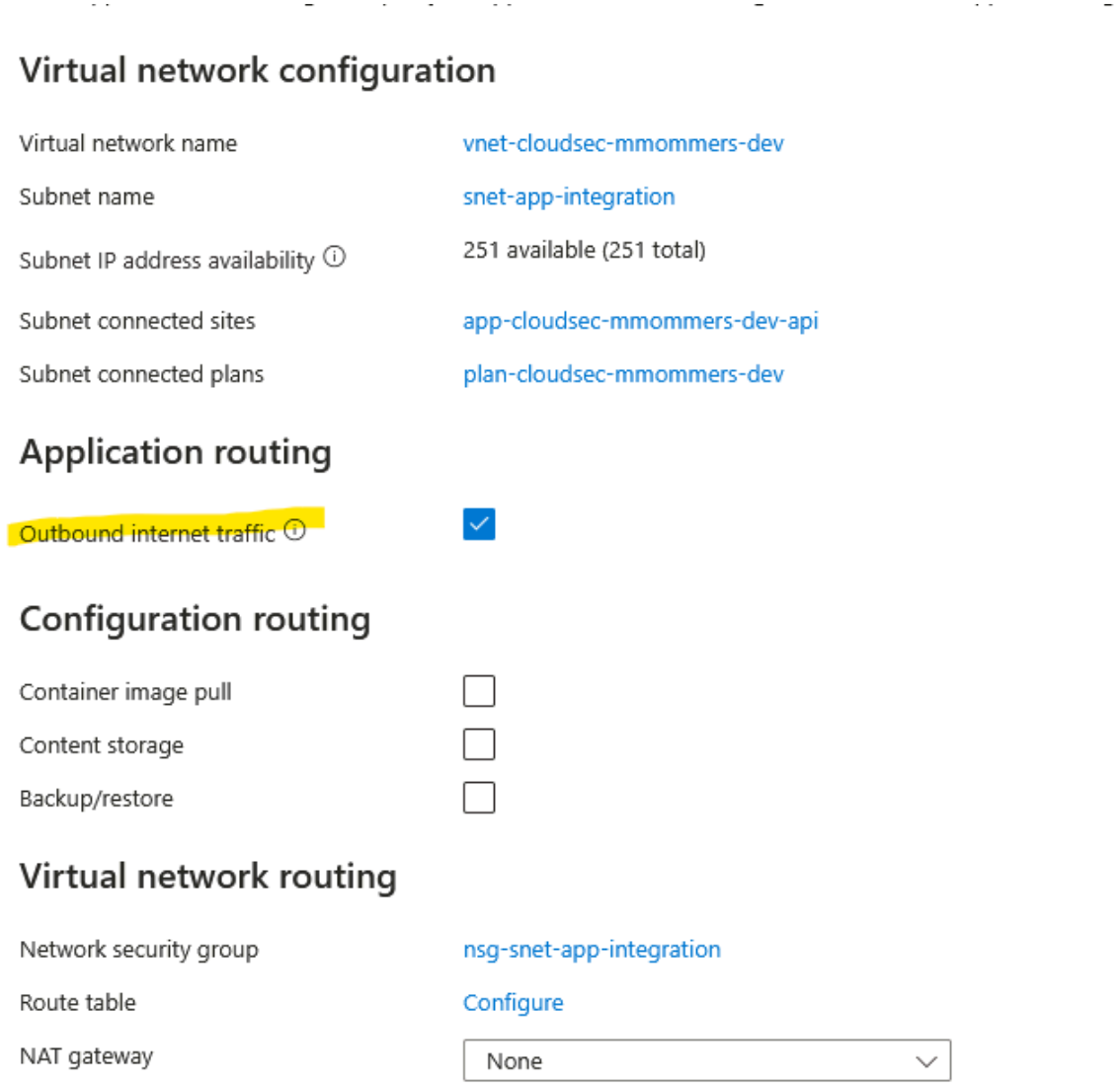


Write down the result:

Item	Value
VNet name	
Subnet name	

4.2. Verify WEBSITE_VNET_ROUTE_ALL setting

1. Go to the VNet configuration
2. Make sure “Outbound internet traffic” is checked



> **Note**
>
> This setting is critical: it ensures ALL outbound traffic (including calls to public endpoints like `*.azurewebsites.net`) routes through the VNet integration subnet.
>
> Without this setting, only RFC 1918 private IP traffic (10.x.x.x, 172.16.x.x, 192.168.x.x) goes through the VNet. Public endpoint calls would bypass the VNet integration, and the access restriction would block them.

4.3. Test direct call to internal backend (before the fix)

Before adding the access restriction, confirm the direct call currently works:

```
curl https://<INTERNAL_BACKEND_URL>/internal/health
```

Expected result before the fix:

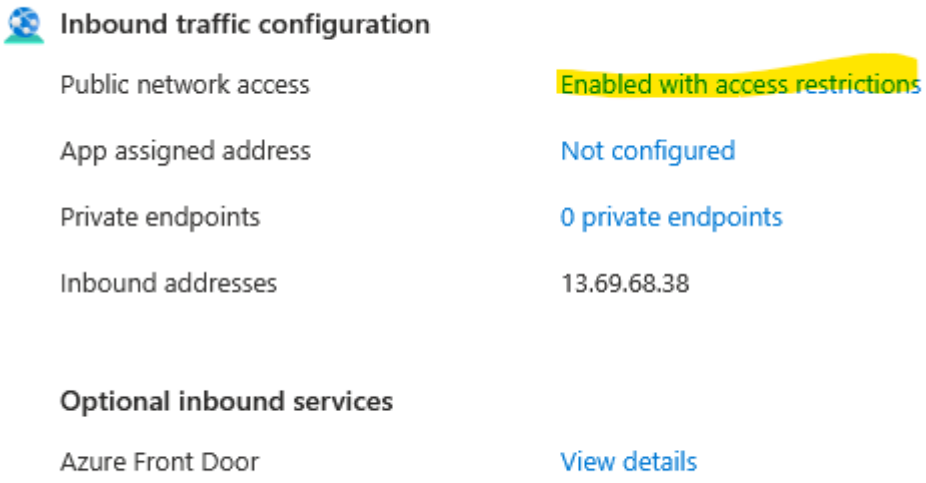
```
HTTP 200 — service responds
```

Write down the actual HTTP status and response:

Item	Value
HTTP status	
Response body	

5. Part 2 — Add an access restriction to the internal backend

5.1. Azure Portal



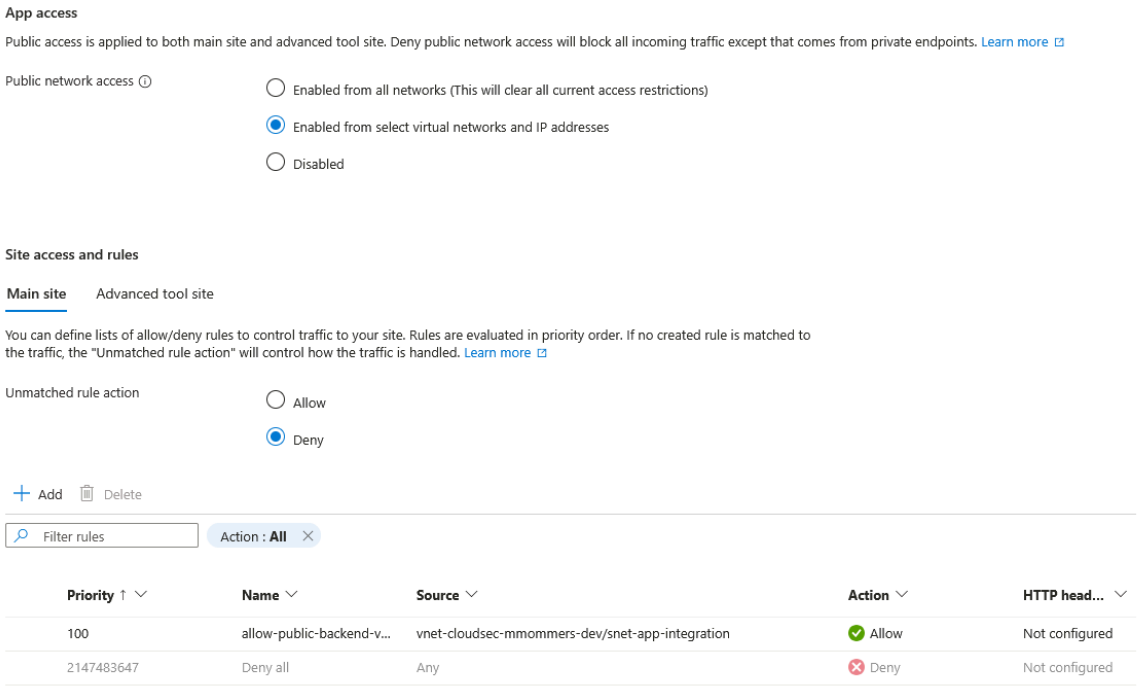
1. Open the Azure Portal and navigate to your internal backend App Service.
2. Go to **Settings → Networking → Inbound traffic → Public Network Access**.
3. Public Network Access -> Enabled from select virtual networks and IP addresses
4. Site Access and Rules -> Main Site
5. Click **+ Add**.
6. Configure the rule:

Setting	Value
Name	`allow-public-backend-vnet`
Action	Allow
Priority	`100`
Type	Virtual network
Virtual network	Select your VNet
Subnet	`snet-app-integration`

7. Click **Add rule**.
8. Under **Unmatched rule action**, select **Deny**.
9. Click **Save**.
10. Wait 30-60 seconds for the change to propagate.

> **Note**

> The access restriction allows traffic from the VNet integration subnet. Because the public backend has `Outbound internet traffic` enabled, its calls to the internal backend route through that subnet and are allowed. Direct calls from the internet are blocked.



Write down any issues encountered:

Your answer:

6. Part 3 — Test after the fix

6.1. Direct call (should be blocked)

```
curl -v https://<INTERNAL_BACKEND_URL>/internal/health
```

Expected result after the fix:

```
HTTP 403 Forbidden — or connection refused
```

Write down the actual result:

Item	Value
HTTP status	
Response body	

6.2. Path through public backend (should still work)

```
curl https://<PUBLIC_BACKEND_URL>/api/network-check
```

> **Note**
>
> The `/api/network-check` endpoint on the public backend makes a call to the internal backend's `/internal/health` endpoint. If the access restriction is configured correctly, this call should succeed because it comes from the VNet integration subnet.
>
> **Cold Start warning**: If the public backend app has been idle for a while, the first request may take 20-30 seconds due to cold start. If it times out, wait a minute and try again.

Expected result:

```
{ "reachable": true }
```

Write down the actual result:

Item	Value
`reachable` field	
HTTP status	

> **Note**
>
> Troubleshooting: If the public backend cannot reach the internal backend after adding the access restriction:
>
> 1. Missing `WEBSITE_VNET_ROUTE_ALL=1`: Verify the public backend has this app setting (Settings → Environment variables → App settings). Without it, calls to public endpoints (*.azurewebsites.net) bypass the VNet integration.
>   - Fix: Add the app setting, save, and restart the App Service
> 2. Wrong subnet: Verify the access restriction allows the `snet-app-integration` subnet (the VNet integration subnet)

3. Default action not Deny: Verify the unmatched rule action is set to Deny
4. Propagation delay: Access restriction changes can take 30-60 seconds to apply. Wait and retry.

6.3. Full trace demo (should still work end to end)

```
curl https://<PUBLIC_BACKEND_URL>/api/trace-demo
```

Does the trace demo complete successfully?

Your answer:

Write down the correlation ID from this successful request:

Your answer:

7. Part 4 — Compare before and after

Complete the table based on your test results.

Call	Before fix	After fix
Direct call to internal backend	HTTP 200	
Public backend → internal backend	HTTP 200	
Full trace demo via public backend	Works	

8. Reflection questions

8.1. Question 1

Why is blocking everything not the correct goal?

Your answer:

8.2. Question 2

What is the difference between an App Service access restriction and an NSG rule in terms of where the enforcement happens?

Your answer:

8.3. Question 3

Could an attacker bypass this restriction by using the public backend as a proxy?

Your answer:

9. Final conclusion

At the end of this lab, your conclusion should look similar to this:

```
The access restriction blocks all inbound traffic to the internal
backend except from the VNet integration subnet used by the public backend.

Direct calls from the public internet now receive HTTP 403.

The expected path through the public backend continues to work.
The application behavior has not changed for normal users.

Only the attack surface changed.
```