THOMAS MORE

Optional Lab 2.x

Application Gateway and WAF

Toepasbare Cloud Security voor IT-Professionals

2025 - 2026

Contents

1. Application Gateway and WAF

1.1. Context

In the core Day 2 labs, you restricted access to the internal backend and moved the DB storage account and Key Vault behind private endpoints.

The public backend API is still reachable directly from the internet. Any HTTP request can reach the application code — including malicious ones.

In this lab you will deploy an Azure Application Gateway with a Web Application Firewall (WAF) in front of the public backend. The WAF inspects every request against the OWASP Core Rule Set and blocks known attack patterns before they reach the application.

You will then add an access restriction so the public backend only accepts traffic through the Application Gateway, completing the defence-in-depth chain.

1.2. Learning goals

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

- What Application Gateway is and where it sits in the architecture.
- What a WAF does and what the OWASP Core Rule Set protects against.
- The difference between WAF Detection mode and Prevention mode.
- Why the App Service still needs an access restriction after App Gateway is deployed.
- How the App Gateway pattern mirrors the Lab 2.3 access restriction pattern one layer up.

1.3. Estimated time

65 minutes

This lab is optional and instructor-led.

Suggested timing:

Time	Activity
5 minutes	Micro-theory and context
20 minutes	Create App Gateway WAF v2 in the Portal
10 minutes	Test traffic through the App Gateway
5 minutes	Configure diagnostic settings for WAF logs
10 minutes	Trigger a WAF block and observe the log
10 minutes	Add access restriction to lock down the App Service
5 minutes	Reflection questions

> **Note**
>
> Application Gateway WAF v2 costs approximately $0.45 per hour. Create it for this lab only. Delete it from the Portal when done.

1.4. Before you start

Make sure you have:

- [ ] The `public_backend_url` from earlier labs
- [ ] Azure Portal access
- [ ] Labs 2.3, 2.4, and 2.5 completed (core architecture is in place)

2. Micro-theory: Application Gateway and WAF

Azure Application Gateway is a regional Layer 7 load balancer. It understands HTTP and can inspect URLs, headers, and request bodies.

The WAF component checks each request against the OWASP Core Rule Set (CRS). The CRS detects common attack patterns including SQL injection, cross-site scripting (XSS), path traversal, and protocol violations.

Component	Role
App Gateway	Receives traffic from the internet, routes to backend
WAF policy	Inspects each request against OWASP CRS rules
Backend pool	The App Service(s) that receive forwarded requests
Health probe	Checks that the backend is responding before forwarding

The App Gateway in this lab listens on HTTP port 80. The backend connection to the App Service uses HTTPS. In production you would configure an HTTPS listener with a certificate from Key Vault.

> **Note**
>
> A WAF is not a substitute for input validation in the application. It is a complementary layer. An application should still validate and sanitise all input.

3. Part 1 — Create the App Gateway WAF v2

The `snet-appgw` subnet (`10.0.4.0/24`) is already pre-provisioned in your VNet by Terraform.

3.0.1. Step 1 — Create a WAF policy

1. In the Azure Portal, search for **Web application firewall policies** and click **+ Create**.
2. Fill in the **Basics** tab:

Setting	Value
Policy for	Regional WAF (Application Gateway)
Subscription	Your subscription
Resource group	Your lab resource group
Name	`wafpol-<your-prefix>`
Region	Same region as your resources

3. Set **Policy mode** to **Prevention**.
4. On the **Managed rules** tab, confirm OWASP CRS 3.2 is selected.
5. Skip **Custom rules** and **Association**.
6. Click **Review + create**, then **Create**.

3.0.2. Step 2 — Create the Application Gateway

1. Search for **Application gateways** and click **+ Create** choose the regular option (non-containers).
2. Fill in the **Basics** tab:

Setting	Value
Resource group	Your lab resource group
Name	`agw-<your-prefix>`
Region	Same region as your resources
Tier	WAF V2
Minimum instance count	`1`
Maximum instance count	`2`
Virtual network	Your lab VNet
Subnet	`snet-appgw`

3. On the **Frontends** tab:
   - Frontend IP address type: **Public**
   - Create a new public IP address, name it `pip-agw-<your-prefix>`
4. On the **Backends** tab:
   - Click **+ Add a backend pool**
   - Name: `pool-public-backend`
   - Add target: **App Services** → select your public backend App Service
5. On the **Configuration** tab, click **+ Add a routing rule**:
   - Rule name: `rule-http-to-backend`

- Priority: `100`
- **Listener** sub-tab:
  - Listener name: `listener-http`
  - Frontend IP: Public
  - Protocol: HTTP
  - Port: 80
- **Backend targets** sub-tab:
  - Target type: Backend pool ⟶ `pool-public-backend`
  - Backend settings: click **Add new**
    - Name: `settings-public-backend`
    - Backend protocol: HTTPS
    - Backend port: 443
    - Override with new host name: **Yes — pick host name from backend target**
    - Create custom health probe: Yes

7. Click **Next**, then **Review + Create -> Create**.

> **Note**
>
> App Gateway provisioning takes 5-10 minutes. This is normal.

Write down any issues encountered during creation:

Your answer:

4. Part 2 — Verify provisioning

4.1. Collect the App Gateway public IP

Navigate to your Application Gateway in the Azure Portal.

Go to **Overview** and find the **Frontend public IP address**.

Write down the IP address:

Field	Value
App Gateway public IP	
App Gateway name	

4.2. Check the backend health probe in the Portal

1. Navigate to your Application Gateway in the Azure Portal.
2. Go to **Monitoring → Backend health**.
3. Confirm the backend pool shows the public backend App Service as **Healthy**.

Write down the backend health status:

Your answer:

If the backend shows **Unhealthy**, check that the `/api/health` endpoint returns HTTP 200:

```
curl https://<PUBLIC_BACKEND_URL>/api/health
```

5. Part 3 — Test traffic through the App Gateway

Call the health endpoint through the App Gateway (HTTP, port 80):

```
curl http://<APPGW_PUBLIC_IP>/api/health
```

Write down the result:

Item	Value
HTTP status	
Response body	

Call the trace demo through the App Gateway:

```
curl http://<APPGW_PUBLIC_IP>/api/trace-demo
```

Write down the correlation ID from the response:

Your answer:

5.1. Compare direct App Service call vs App Gateway call

At this point, both paths still work. The App Service URL is not yet restricted.

Path	Expected result	Actual result
Direct: `https://<PUBLIC_BACKEND_URL>/api/health`	HTTP 200	
Via App Gateway: `http://<APPGW_IP>/api/health`	HTTP 200	

6. Part 4 — Configure diagnostic settings

Before you can query WAF logs, you need to send them to a Log Analytics workspace.

6.1. Create diagnostic settings

1. Navigate to your Application Gateway in the Azure Portal.
2. Go to **Monitoring → Diagnostic settings**.
3. Click **+ Add diagnostic setting**.
4. Fill in the settings:

Setting	Value
Diagnostic setting name	`diag-appgw-logs`
Logs → Categories	Select **ApplicationGatewayFirewallLog**
Destination details	**Send to Log Analytics workspace**
Log Analytics workspace	Select the workspace from your resource group (created by Terraform)

5. Click **Save**.

> **Note**
>
> Logs start flowing to the workspace immediately, but there is a 2-5 minute ingestion delay before they appear in queries.

Write down any issues encountered:

Your answer:

7. Part 5 — Trigger a WAF block

The WAF is in Prevention mode with the OWASP CRS 3.2 ruleset. Send a request with a SQL injection pattern in the query string.

```
curl -v "http://<APPGW_PUBLIC_IP>/api/health?id=1' OR '1'='1"
```

Write down the HTTP response code:

Item	Value
HTTP status	
Response body or error	

The WAF should return HTTP 403 with a block message.

7.1. View the WAF log in the Portal

Wait 2-5 minutes after the blocked request, then query the logs:

1. Navigate to your Application Gateway → **Monitoring → Logs**.
2. Close the Queries helper dialog if it appears.
3. Run the following query to find the blocked request:

```
AGWFirewallLogs
| where Action == "Matched"
| order by TimeGenerated desc
| take 10
```

Write down the rule that matched:

Item	Value
Rule ID (RuleId)	
Message (Message)	
Action (Action)	

> **Note**
>
> If the query returns no results, wait another 2-3 minutes and retry. Diagnostic logs have a small ingestion delay. If logs still don't appear after 5 minutes, verify diagnostic settings are configured and saved.

8. Part 6 — Add access restriction to the public backend

The App Gateway is working. Now lock down the public backend so it only accepts traffic from the App Gateway subnet.

This is the same pattern from Lab 2.3 — but one layer up in the architecture.

8.1. In the Azure Portal

1. Navigate to your public backend App Service.
2. Go to **Settings → Networking → Access restrictions**.
3. Add a new inbound rule:

Setting	Value
Name	`allow-appgw-subnet`
Action	Allow
Priority	`100`
Source	Virtual Network — select `snet-appgw`

4. Set the default action (unmatched rules) to **Deny**.
5. Save.

Write down any issues encountered:

Your answer:

8.2. Test after the restriction

Direct call to the App Service (should be blocked):

```
curl -v https://<PUBLIC_BACKEND_URL>/api/health
```

Write down the result:

Item	Value
HTTP status	
Direct access blocked?	

Call via App Gateway (should still work):

```
curl http://<APPGW_PUBLIC_IP>/api/health
```

Write down the result:

Item	Value

HTTP status	
Response body	

9. Part 7 — Full architecture verification

Complete the table to confirm the end-to-end architecture is working:

Path	Expected	Actual
Direct: `https://<PUBLIC_BACKEND_URL>/api/health`	403	
Via App Gateway: `/api/health`	200	
Via App Gateway: `/api/trace-demo`	200	
WAF SQL injection test via App Gateway	403	
Internal backend direct call	403 (from Lab 2.3)	

10. Reflection questions

10.1. Question 1

In Lab 2.3, you added an access restriction to the internal backend so only the public backend can call it. In this lab, you added an access restriction to the public backend so only the App Gateway can call it. What does this chain prevent?

Your answer:

10.2. Question 2

The WAF blocked a SQL injection in the query string. Does that mean the application code no longer needs to validate input?

Your answer:

10.3. Question 3

A WAF in Detection mode logs matching requests but does not block them. When would you use Detection mode instead of Prevention mode?

Your answer:

10.4. Question 4

The App Gateway listener in this lab uses HTTP (port 80). What would you need to add to serve HTTPS on port 443 instead?

Your answer:

10.5. Question 5

Describe the full request path from a user's browser to the DB storage account after all Day 2 controls are in place.

Your answer:

11. Final conclusion

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

```
The Application Gateway WAF v2 sits in front of the public backend API.
Every HTTP request is inspected against the OWASP Core Rule Set.
SQL injection and XSS patterns are blocked before they reach the application.

The public backend App Service now has an access restriction that blocks
direct access from the internet. Only the App Gateway subnet can reach it.

The defence-in-depth chain is complete:

Internet → App Gateway WAF → Public Backend (restricted)
        → Internal Backend (restricted to Public Backend subnet)
        → DB storage account (private endpoint, public access disabled)
        → Key Vault (private endpoint, public access disabled)

Each layer removes one attack surface.
No single layer is sufficient on its own.
```

> **Note**
>
> Delete the App Gateway after the lab: navigate to the Application Gateway resource in the Azure Portal and click Delete. Leaving it running costs approximately $0.45 per hour.