THOMAS MORE

Lab 4.2

Verifying Platform Logs in Log Analytics

Toepasbare Cloud Security voor IT-Professionals

2025 - 2026

Contents

1. Verifying Platform Logs in Log Analytics

1.1. Context

The Day 4 infrastructure was deployed with diagnostic settings configured via Terraform (Lab 4.1). These settings route platform logs from App Services, Key Vault, and Storage to your Log Analytics workspace. In this lab, you verify that those diagnostics are working correctly, explore the available log tables, and learn how to set the daily ingestion cap in the Azure portal. You will also compare platform-level logs (Diagnostic Settings) with application-level logs (OpenTelemetry-based SDK).

By the end of this lab, you will understand which Azure resources produce which types of logs, how diagnostic settings route those logs to Log Analytics, and how to query them using KQL.

1.2. Learning goals

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

- The three types of Azure Monitor data: logs, metrics, and traces.
- How diagnostic settings route platform logs from App Services, Key Vault, and Storage to Log Analytics.
- The difference between OpenTelemetry-based SDK data (`AppRequests`, `AppDependencies`, `AppTraces`, `AppExceptions`), Diagnostic Settings data (resource-specific tables like `AppServiceHTTPLogs`, `AZKVAuditLogs`), and Activity Logs (`AzureActivity`).
- Which KQL tables contain data from which source. Note: `AzureActivity` is streamed automatically to the Log Analytics workspace; no Sentinel connector is required.
- How to set the daily cap on Log Analytics in the Azure portal to prevent runaway costs.
- How to query platform-level logs using KQL in Log Analytics.

1.3. Estimated time

85 minutes

Suggested timing:

Time	Activity
10 minutes	Generate initial traffic to the backend API
10 minutes	Review the Log Analytics workspace and available tables
10 minutes	Set the daily cap on Log Analytics in the Azure portal
20 minutes	Query resource-specific tables for platform-level logs (App Service, Key Vault, Storage)
15 minutes	Query SDK-generated logs (AppRequests, AppTraces, AppExceptions)
15 minutes	Generate activity log events and query AzureActivity
10 minutes	Compare SDK data vs. platform data
15 minutes	Reflection and discussion

1.4. Before you start

Make sure you have:

- [ ] Day 4 pipeline has run and deployed monitoring infrastructure (Lab 4.1)
- [ ] You have access to the Log Analytics workspace in the Azure portal
- [ ] Owner or Contributor role on the resource group

2. Generate Initial Traffic

Before querying logs, you need traffic hitting your backend so that App Service HTTP logs, OpenTelemetry-based SDK data, and Key Vault audit events are generated.

1. Find the URL of your public backend App Service in the Azure portal (e.g., `https://app-<your_prefix>-api-d4.azurewebsites.net`)
2. Run these requests to generate baseline telemetry:

```
# Generate legitimate traffic
curl https://<your-backend-url>/api/health
curl https://<your-backend-url>/api/public-message
curl https://<your-backend-url>/api/config

# Generate trace traffic with correlation ID
curl -H "x-correlation-id: lab-4-2-test" https://<your-backend-url>/api/trace-demo

# Generate Key Vault access traffic
curl https://<your-backend-url>/api/security/secret-demo
```

3. Wait about 60 seconds for the logs to propagate to Log Analytics.

2.1. Verify Data Is Present

Run a quick check in Log Analytics to confirm data is flowing:

```
AppServiceHTTPLogs
| take 5
```

If you see results, you're ready to proceed. If not, wait another minute and try again.

3. Review the Log Analytics Workspace

3.1. Navigate to Log Analytics

1. In the Azure portal, search for **Log Analytics workspaces**
2. Click on your workspace: `law-<your_prefix>-dev-d4`
3. In the left menu, click **Logs** (under General)

3.2. Discover Available Tables

In the Logs query editor, click **Tables** on the right side. You should see:

Table	Source	Description
`AppRequests`	OpenTelemetry-based SDK	HTTP requests captured by the SDK
`AppDependencies`	OpenTelemetry-based SDK	Database/API calls made by the app
`AppTraces`	OpenTelemetry-based SDK	Custom log messages
`AppExceptions`	OpenTelemetry-based SDK	Error stack traces
`AppServiceHTTPLogs`	Diagnostic Settings (resource-specific)	App Service HTTP logs
`StorageBlobLogs`	Diagnostic Settings (resource-specific)	Blob storage operations
`AzureActivity`	Azure Activity Log	Management plane operations

> **Note**
>
> The `AzureActivity` table may be empty at first — the Azure Activity Log streams to the workspace automatically, but data propagation typically takes 15–30 minutes after an event occurs.

> **Question**
>
> Why are there multiple tables for the same resource? For example, why does the App Service produce data in both the `AppRequests` table and the `AppServiceHTTPLogs` table?

4. Set the Daily Cap

4.1. Configure the Daily Ingestion Cap in the Portal

Log Analytics charges by data ingestion volume. To prevent runaway costs, you should set a daily cap. This is done in the Azure portal.

1. In the Log Analytics workspace, click **Settings** in the left menu, then **Usage**
2. Look for **Daily cap** (or **Quota**) in the settings list
3. Set the daily cap to **1 GB**

> **Note**
>
> When the daily cap is reached, *data collection stops* for the rest of the day — not just new entries, but all ingestion into billable tables. This protects you from unexpected charges but also means you lose visibility into your environment until the next reset cycle. For lab purposes, 1 GB is more than enough to generate and query telemetry.

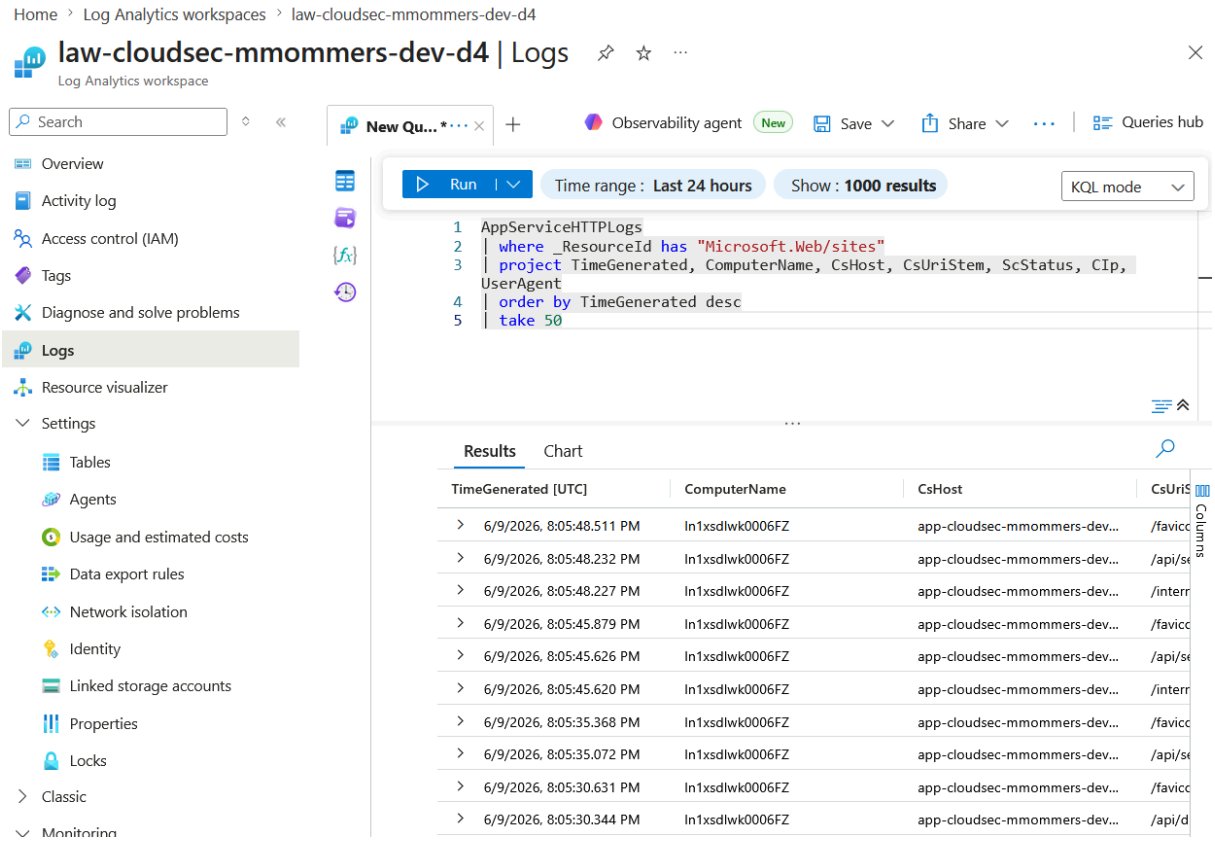
> **Question**
>
> Why is setting the daily cap a manual step in the Azure portal rather than something you configure in Terraform? What operational skill does this teach you?

5. Query Platform-Level Logs

5.1. Query App Service HTTP Logs (resource-specific table)

Modern Azure services stream logs to dedicated resource-specific tables instead of the legacy `AzureDiagnostics` table. Query the `AppServiceHTTPLogs` table directly:

```
AppServiceHTTPLogs
| where _ResourceId has "Microsoft.Web/sites"
| project TimeGenerated, ComputerName, CsHost, CsUriStem, ScStatus, CIp, UserAgent
| order by TimeGenerated desc
| take 50
```



You should see rows with columns for the request details. Look for:

- `CsHost` — the App Service instance domain (e.g., `app-cloudsec-...-api-d4...`)
- `ScStatus` — HTTP status codes (200, 404, 500, etc.)
- `CsUriStem` — the URL path requested
- `CIp` — the IP address of the client making the request
- `UserAgent` — the user agent string from the HTTP request

> **Question**
>
> What is the difference between the data in the `AppRequests` table (OpenTelemetry-based SDK) and the data in the `AppServiceHTTPLogs` table (Diagnostic Settings)? List at least three differences.

5.2. Query SDK-Generated Logs

The OpenTelemetry-based SDK (`@azure/monitor-opentelemetry`) automatically captures HTTP requests, dependencies (database calls), traces, and exceptions. Use these queries as a starting point for your own investigations:

```
// Recent HTTP requests with response codes
AppRequests
| project Name, Success, DurationMs, ClientIP, TimeGenerated
| where TimeGenerated > ago(1h)
| order by TimeGenerated desc
| take 20

// Failed requests (status >= 400)
AppRequests
| where TimeGenerated > ago(1h) and not(Success)
| project TimeGenerated, Name, DurationMs, ClientIP
| order by TimeGenerated desc

// Slowest operations
AppRequests
| where TimeGenerated > ago(1h)
| project TimeGenerated, Name, DurationMs, Success, ClientIP
| order by DurationMs desc
| take 10
```

> **Question**
>
> What additional information does the SDK `AppRequests` table provide that the `AppServiceHTTPLogs` table does not? Try comparing the columns in both tables.

5.3. Add Custom Trace Logs to Your Application

The SDK also captures custom trace logs from your application code. To see this in action, add a trace log to one of your API endpoints. Open `backend/src/routes/securityLabRoutes.ts` and find the `/api/security/secret-demo` route. Add a trace log based on the existing logs.

Deploy the change and call the endpoint again:

```
curl https://<your-backend-url>/api/security/secret-demo
```

Then query the `AppTraces` table to see your custom log:

```
AppTraces
| where TimeGenerated > ago(5m)
| project Message, TimeGenerated, Properties
| order by TimeGenerated desc
```

> **Question**
>
> What is the difference between a `trace`, a `request`, and a `dependency` in Application Insights? When would you use each?

5.4. How the SDK Captures Dependencies Across Services

When your public backend calls the internal backend, Key Vault, or Table Storage, the SDK captures those calls as dependencies in the `AppDependencies` table with an `OperationId`. This links them back to the original request.

To see this in action, make a call that triggers internal service calls:

```
curl https://<your-backend-url>/api/network-check
```

The public backend forwards this request to the internal backend, passing the `correlationId` header. In Application Insights, you can find these dependencies by querying the `AppDependencies` table for your target services (e.g., `Target contains "vault.azure.net"`).

> **Note**
>
> Tracing a request through the full stack — from the public backend to the internal backend to Key Vault and Table Storage — is covered in Lab 4.3, where you'll learn how to join AppRequests, AppDependencies, and AppTraces using `correlationId`.

5.5. Generate Activity Log Events

To see activity logs in action, perform some operations in the Azure portal. Each action generates an entry in the Azure Activity Log that is streamed to your Log Analytics workspace.

1. Navigate to your public backend App Service (`app-<your_prefix>-api-d4`) and click **Restart** — confirm the restart.
2. Navigate to your Key Vault (`kv-<your_prefix>-d4`) and click **Access policies** → **Create** — you'll see an audit event.
3. Navigate to your storage account and create a blob container: go to **Containers** → **+ Container**, enter a name like `test`, and click **Create**.

Wait about 60 seconds for the logs to propagate, then run this query:

```
AzureActivity
| where TimeGenerated > ago(5m)
| project TimeGenerated, OperationNameValue, Caller, ResourceGroup, SubscriptionId
| order by TimeGenerated desc
```

You should see entries like `Microsoft.Web/sites/restart`, `Microsoft.KeyVault/vaults/write`, and `Microsoft.Storage/storageAccounts/blobServices/containers/write`.

> **Question**
>
> What is the difference between the data in `AzureActivity` and the data in `AppServiceHTTPLogs` (or other resource-specific tables)? Why do both exist?

5.6. Compare SDK vs. Platform Data

Run this query to compare OpenTelemetry-based SDK data with Diagnostic Settings data for the same time period:

```
let lookback = 1h;
let sdk_requests = AppRequests
    | where TimeGenerated > ago(lookback)
    | project
        Source = "Application Insights SDK",
        TimeGenerated,
        Request = Name,
        Method = "",
        Path = tostring(parse_url(Url).Path),
        StatusCode = toint(ResultCode),
        Success = tobool(Success),
        DurationMs = todouble(DurationMs),
        ClientIP = tostring(ClientIP),
        OperationId = tostring(OperationId);

let diag_requests = AppServiceHTTPLogs
    | where TimeGenerated > ago(lookback)
    | where _ResourceId has "Microsoft.Web/sites"
    | project
        Source = "App Service HTTP logs",
        TimeGenerated,
        Request = CsUriStem,
        Method = tostring(CsMethod),
        Path = tostring(CsUriStem),
        StatusCode = toint(ScStatus),
        Success = iff(toint(ScStatus) between (200 .. 399), true, false),
        DurationMs = todouble(TimeTaken),
        ClientIP = tostring(CIp),
        OperationId = "";

union sdk_requests, diag_requests
| extend
    Severity = case(
        StatusCode >= 500, "High",
        StatusCode in (401, 403), "Medium",
        StatusCode == 404, "Medium",
        StatusCode >= 400, "Low",
        "Info"
    ),
    Lesson = case(
        Source == "Application Insights SDK", "What the app observed",
        Source == "App Service HTTP logs", "What the platform observed",
        "Unknown"
    )
| project
    TimeGenerated,
    Severity,
    Source,
    Lesson,
    Method,
    Path,
    StatusCode,
    Success,
    DurationMs,
    ClientIP,
    OperationId
```

```
| order by TimeGenerated desc
| take 40
```

> **Question**
>
> Compare the SDK AppRequests and Diagnostic AppServiceHTTPLogs in your results. Why might some SDK requests not appear in Diagnostic settings, or vice versa? What additional information does each source provide?

6. Understand Log Propagation

6.1. Generate Fresh Telemetry to Observe Propagation

Run these requests again to generate fresh telemetry and observe how quickly logs propagate:

```
# Additional legitimate traffic
curl https://<your-backend-url>/api/health
curl https://<your-backend-url>/api/public-message

# Additional Key Vault access traffic
curl https://<your-backend-url>/api/security/secret-demo
```

Wait at least 60 seconds for the logs to propagate, then run the AppServiceHTTPLogs query again. How many new rows appeared? What time range do they cover?

6.2. Wait for Log Propagation

Even though diagnostic settings were deployed via Terraform, logs still take time to propagate. After generating fresh traffic, wait 30-120 seconds before querying. If you run queries immediately and see no data, wait a minute and try again.

> **Note**
>
> OpenTelemetry-based SDK data streams near real-time (5-30 seconds). Diagnostic settings data has a longer propagation delay (30-120 seconds). This is because SDK data is pushed directly, while diagnostic settings data is collected by the platform and buffered before being sent to Log Analytics.

7. Reflection Questions

Your answer:

1. The Day 4 infrastructure deployed diagnostic settings on three resources (App Service, Key Vault, Storage). If you needed to detect a security breach where an attacker accessed Key Vault secrets, which table would you query first and why?
2. The SDK `AppRequests` table and the `AppServiceHTTPLogs` table both capture HTTP request data. If you had to choose one for a security investigation, which would you choose and why? Consider the type of information each provides.
3. Why does Diagnostic Settings have a 30-120 second propagation delay while the OpenTelemetry-based SDK is near real-time? What architectural difference causes this?
4. If an attacker modifies the diagnostic settings to disable logging, how would you detect that? What KQL query would you write to check if diagnostic settings are still active?

8. Summary

In this lab, you explored the available log tables in Log Analytics, queried platform-level logs using KQL, queried SDK-generated logs, generated activity log events, and compared OpenTelemetry-based SDK data with Diagnostic Settings data. You now understand the three sources of Azure monitoring data (SDK, Diagnostic Settings, Activity Logs) and how to query them effectively.

8.1. Challenge

Write your own KQL query to find all requests with the same correlation ID in the last hour, joining data from `AppRequests`, `AppDependencies`, and `AppTraces` to reconstruct the full request flow. Use the `OperationId` field to correlate across tables.