Azure Cloud Fundamentals

Lab 3

Monitoring and Alerting for Azure Services

Observing Web App, SQL Database, and Storage with Log Analytics, Application Insights, and Azure Monitor Alerts

Log Analytics · App Insights · Metric Alerts · Log Alerts · Activity Log Alerts · Action Groups

Setting	Value
Module	Monitoring, Logging, and Alerting
Level	Fundamentals (AZ-900 / AZ-104 aligned)
Estimated time	120 – 180 minutes
Estimated cost	< €1 if the lab is completed and cleaned up the same day
Region	West Europe
Prerequisite	Lab 1 (infrastructure) + Lab 2 (app deployed and running)
Deployment method	Azure Portal (click-through) with optional Azure CLI snippets
Version	1.0 · April 2026

Table of Contents

1. Lab Objective

In Labs 1 and 2 you built the infrastructure and deployed a working application. But right now, if anything breaks, you have no way to know about it — you'd only find out when a user complains. In this lab you will instrument the entire stack with Azure Monitor so that you can observe what's happening and be automatically notified when something goes wrong.

By the end of this lab you will have built:

- **A Log Analytics workspace** that serves as the central store for logs and metrics from all three services
- **Application Insights** auto-instrumented on the Web App (no code changes required)
- **Diagnostic Settings** on the Web App, SQL Database, and Storage Account routing logs to the workspace
- **An Action Group** that sends email notifications to your address when alerts fire
- **8+ metric alerts** covering HTTP errors, response time, CPU, memory, SQL vCore, failed connections, deadlocks, storage availability, and server errors
- **1 log-based (KQL) alert** that queries App Insights for exceptions
- **1 Activity Log alert** that fires when anyone deletes a resource in the resource group
- **A monitoring dashboard** pinning the key metrics in a single view
- **First-hand experience triggering every alert on purpose** so you can see the emails arrive and the alert timeline in the portal

What you will learn

- The difference between metrics, logs, and traces — and when to use each
- How Azure Monitor's data model works:
  data sources → diagnostic settings → destinations
- The three kinds of alert rules in Azure: metric, log, and activity log
- How to write basic KQL (Kusto Query Language) against App Insights and Log Analytics
- How to reason about severity, frequency, and alert fatigue
- The cost levers of monitoring (data ingestion and retention) and how to keep them under control

> **NOTE**
> This lab assumes Labs 1 and 2 are complete and that the web app at https://my-app-001.azurewebsites.net is rendering the student table. If it is not, go back and finish Lab 2 before starting this lab — you need traffic to generate for the alerts to fire.

2. Why Monitoring Matters

Before diving into the portal, it's worth slowing down for a moment. Monitoring is one of those topics that sounds dry on paper but is the difference between a calm on-call shift and a 3 AM wake-up call. Three ideas drive everything in this lab.

2.1 You can't fix what you can't see

Your application and infrastructure emit an enormous amount of telemetry every second — request rates, error counts, dependency latencies, CPU usage, database waits, blob transactions, login failures. By default, most of this is either discarded or only retained for a short window in the platform's own dashboards. The first job of monitoring is to capture and persist that telemetry in a place where you can query it days, weeks, or months later.

2.2 Alerts are a contract, not a feature

Every alert rule you create is a statement of what 'healthy' means for that service. If you set an alert at "HTTP 5xx > 10 in 5 minutes", you are saying: below that, the app is healthy and I don't need to know; above it, I must know immediately. Badly-tuned alerts produce either silence when things are broken, or a flood of false positives that trains the team to ignore them. Tuning alerts is a skill, and you'll get to practice it in Section 10.

2.3 Alerting is only half the job — dashboards are the other half

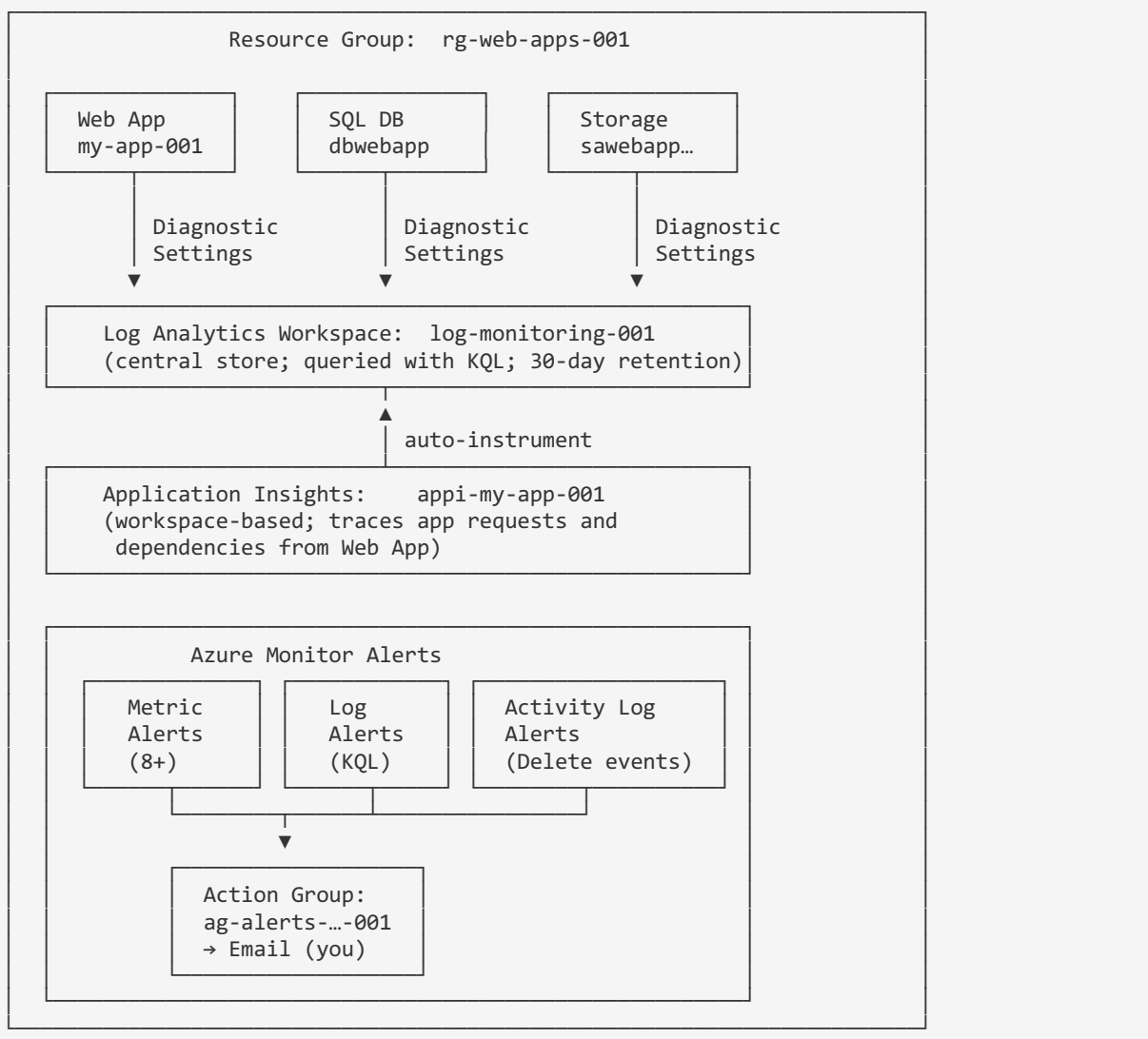
Alerts tell you when something broke. Dashboards tell you what 'normal' looks like so you can spot the drift before it becomes an alert. A 20% increase in average response time may not cross any threshold, but it's visible on a chart, and the team that checks dashboards at the start of every morning catches problems that the alerting system never sees.

2.4 The vocabulary you'll encounter

Term	Meaning
Metric	A numeric value sampled at regular intervals (e.g., CPU % every minute). Cheap to store, fast to query, great for alerts and dashboards.
Log	A timestamped record of an event with arbitrary structured fields (e.g., an HTTP request, an exception, a diagnostic entry). Queryable with KQL, richer than metrics, more expensive to store.
Trace	A chain of related events across services for a single request. Surfaced via Application Insights.
Data source	Where telemetry comes from (a resource's platform telemetry, app code, or the Azure control plane).
Destination	Where telemetry is sent (Log Analytics workspace, Storage Account, Event Hub).
Action Group	A reusable set of notification channels (email, SMS, webhook, function) that one or more alerts can target.
Alert rule	A condition that evaluates metrics or logs on a schedule and triggers the action group when true.
Signal	The specific metric or log query an alert evaluates (e.g., Http5xx, Deadlocks).
Severity	Sev 0 (critical) to Sev 4 (informational) — used to route and prioritize alerts.

3. Architecture and the Azure Monitor Data Model

Azure Monitor is not a single service — it's an umbrella over several components that together provide the monitoring, logging, and alerting capabilities. The diagram below shows the components you will wire up in this lab and the direction telemetry flows.



3.1 How telemetry gets from a resource to an alert

The flow is always the same three steps, regardless of the resource:

- **Step 1 — Emit.** Every Azure resource emits two kinds of telemetry by default: platform metrics (always free, retained for 93 days) and the Azure Activity Log (control-plane events like create/update/delete).
- **Step 2 — Route.** Diagnostic Settings are what you configure to route resource logs (not metrics — those are automatic) to one or more destinations: a Log Analytics workspace, a Storage Account, or an Event Hub.
- **Step 3 — Consume.** Dashboards, Workbooks, and Alert Rules read from the workspace (for logs) or directly from the metrics API (for metrics) and notify you via Action Groups.

3.2 Metric alerts vs. log alerts — when to use which

	Metric alerts	Log alerts
Data source	Platform metrics API (native)	Log Analytics or App Insights (via KQL)
Latency	1–5 minutes to fire	5–15 minutes to fire
Cost per rule	~€0.10/month (very cheap)	~€0.50/month + query execution cost
Good for	CPU, memory, request count, error count, size	Complex conditions, multiple resources, correlation
Example	"CPU % > 80 for 5 minutes"	"3+ exceptions of the same type in 10 min from App Insights"

> 💡 **TIP**
> The rule of thumb: if you can express the condition as a single number over time, use a metric alert. If you need to filter, aggregate across dimensions, or correlate — use a log alert.

4. Prerequisites

Before starting this lab, confirm the following:

4.1 From Labs 1 and 2

- The resource group `rg-web-apps-001` exists with all five resources
- The Web App `my-app-001` (**the-name-of-your-webapp**) is deployed and **actually serving the Student table** (not the default placeholder page)
- You can open `https://<my-app-001>.azurewebsites.net` and see data
- You have Contributor or Owner role on the resource group (Monitoring Contributor is enough for most tasks, but Contributor is simpler)

4.2 New for this lab

- A working email address you will receive alerts on (your student / personal email is fine)
- (Optional) Azure CLI installed, if you prefer mixing CLI with portal
- (Optional) A small HTTP load-testing tool for Section 14. **curl** or PowerShell's `Invoke-WebRequest` in a loop is enough — no need for a dedicated tool.

> ⚠ **WARNING**
> You will configure alerts that send email to the address you specify. If you use a corporate email, make sure your IT is OK with automated email from azure-noreply@microsoft.com. Otherwise, use a personal address for the lab.

5. Naming Conventions and Resources Summary

We continue the CAF abbreviations from Lab 1. All new resources are created in the existing rg-web-apps-001 resource group in West Europe.

Resource Type	CAF Abbreviation	Name for this lab
Log Analytics Workspace	log	log-monitoring-001
Application Insights	appi	appi-my-app-001
Action Group	ag	ag-alerts-email-001
Metric Alert rules	al	al-<resource>-<signal>-001 (e.g., al-webapp-5xx-001)
Log (KQL) Alert rule	al	al-appi-exceptions-001
Activity Log Alert	al	al-rg-resource-delete-001
Dashboard	dash	dash-monitoring-001

> **TIP**
> Consistent alert naming matters more than you'd think. When your team wakes up at 2 AM to a page titled "al-webapp-5xx-001 triggered", they immediately know: it's the Web App, it's about 5xx errors, it's the first rule of that type. Compare with "High error rate" or "Alert 3" — no context, more time wasted.

6. Create the Log Analytics Workspace

The Log Analytics workspace is the central repository where all the diagnostic logs from your other resources will land. It's queryable with KQL (Kusto Query Language), which is the same query language used by Microsoft Sentinel, Defender, and most Microsoft observability tooling.

1. In the Azure Portal, search for **Log Analytics workspaces** and select it.
2. Click **+ Create**.
3. On the Basics tab, enter:
    - **Subscription:** your lab subscription
    - **Resource group:** `rg-web-apps-001`
    - **Name:** `log-monitoring-001`
    - **Region:** West Europe
4. On the **Pricing tier** tab, leave the default **Pay-as-you-go (Per GB 2018)**. This is the standard tier — the first 5 GB per billing account per month is free.
5. On the **Tags** tab, optionally add `env=lab` and `owner=<your-name>` (good habit for real work).
6. Click **Review + create**, then **Create**. Deployment takes 30–60 seconds.

6.1 Configure retention

The default log retention is 30 days, which is fine for this lab. For reference, here's what you'd tune in real workloads:

1. Navigate to `log-monitoring-001` → **Settings** → **Usage and estimated costs** → **Data Retention**.
2. Note the default is 30 days. Sliders go from 30 days up to 2 years (730 days).
3. Leave at 30 days for the lab and close the pane.

> **NOTE**
> Retention above 90 days is billed at a discounted rate ('archive tier'). For production, typical choices are: 30 days interactive + 1–2 years archive for compliance, or 90 days interactive for most workloads. The retention is per-table so you can keep noisy tables short and security tables long.

7. Create and Attach Application Insights

Application Insights is the APM (Application Performance Management) layer of Azure Monitor. It captures request rates, exceptions, dependencies, performance counters, and custom events from your application. Critically, you can enable it on an App Service without changing a single line of code — the runtime injects a lightweight agent that observes the app transparently. This is called 'codeless attach' or 'auto-instrumentation'.

7.1 Create the Application Insights resource

1. In the Portal, search for **Application Insights** and select it.
2. Click **+ Create**.
3. On the Basics tab:
    - **Subscription:** your lab subscription
    - **Resource group:** `rg-web-apps-001`
    - **Name:** `appi-my-app-001`
    - **Region:** West Europe
    - **Resource Mode: Workspace-based** (the modern mode — classic is deprecated)
    - **Log Analytics Workspace:** `log-monitoring-001` (the one you just created)
4. Click **Review + create**, then **Create**. Deployment takes ~30 seconds.

> **NOTE**
> Workspace-based App Insights stores its data inside the Log Analytics workspace rather than in a separate store. This means one place to query, one place to set retention, and one bill for data ingestion. Classic (non-workspace) App Insights is scheduled for deprecation — always pick workspace-based for new resources.

7.2 Enable auto-instrumentation on the Web App

Now tell the Web App to send its telemetry to the App Insights resource you just created. On Linux App Service with .NET 10, this happens via a runtime agent and a few App Settings — no code change required.

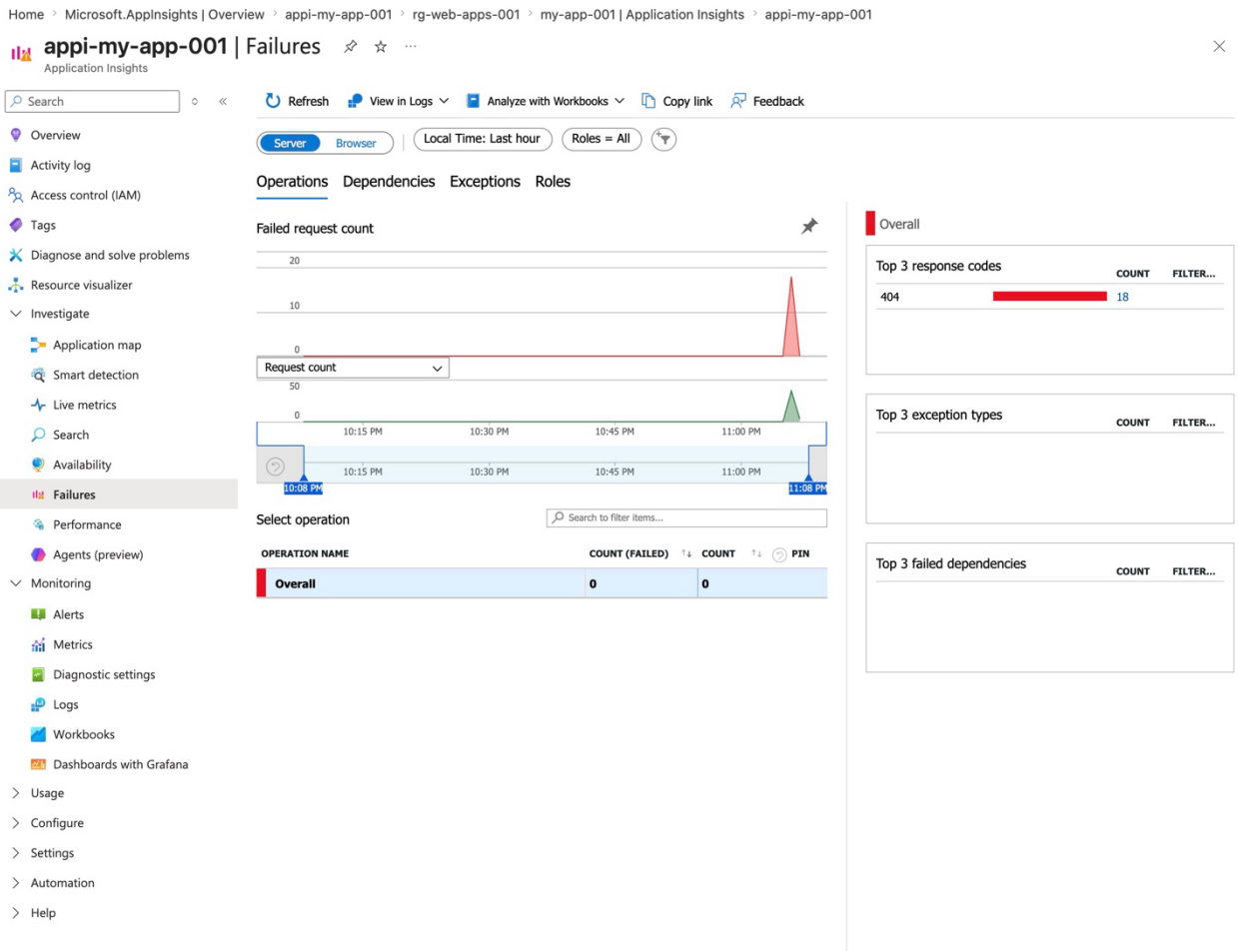
1. Navigate to your Web App `my-app-001`.
2. In the left menu, click **Settings** → **Application Insights**.
3. You'll see the banner **Application Insights is not enabled for this app**. Click **Turn on Application Insights**.
4. In the pane that opens:
    - **Link to an existing Application Insights resource:** choose `appi-my-app-001`
    - **.NET: Recommended** (the default — this enables the runtime attach agent)
    - **Collection level: Recommended**

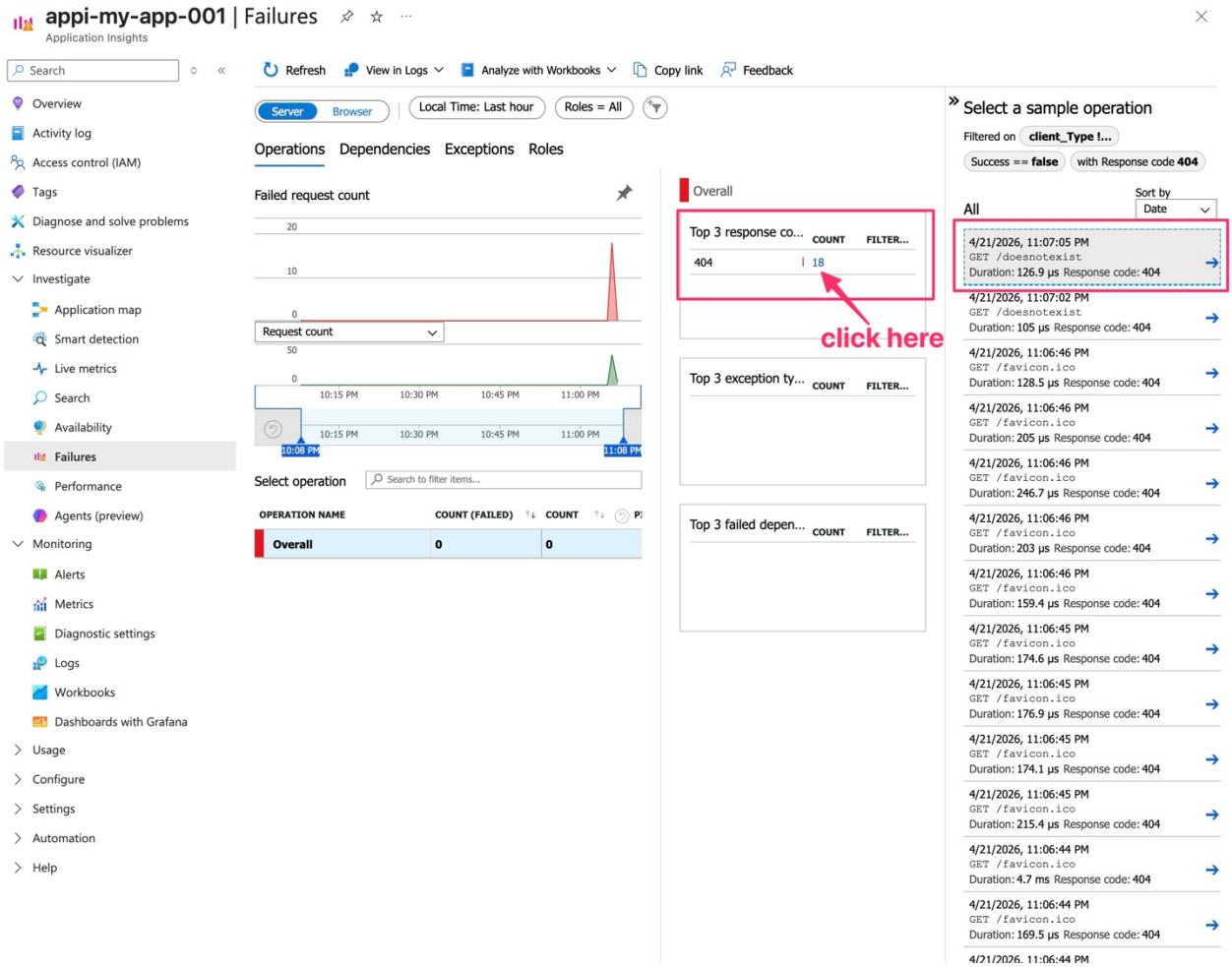
5. Click **Apply** and confirm the **Restart app** prompt.
   The web app will restart — expect 60s downtime.

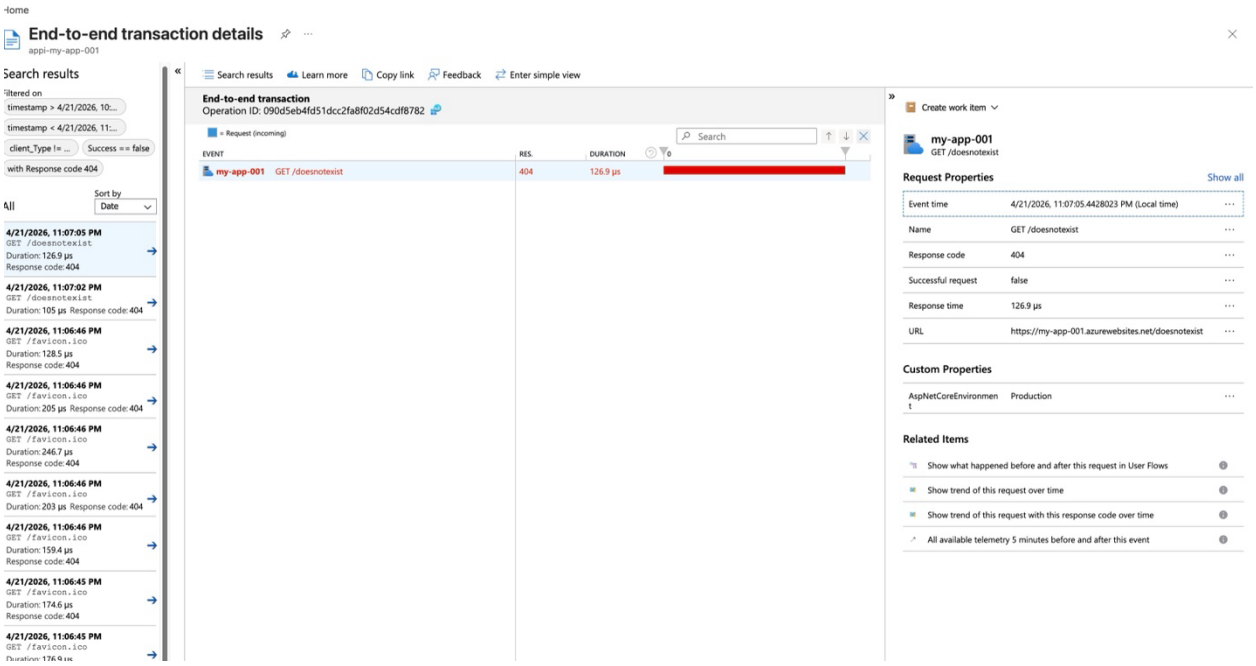
7.3 Generate some telemetry

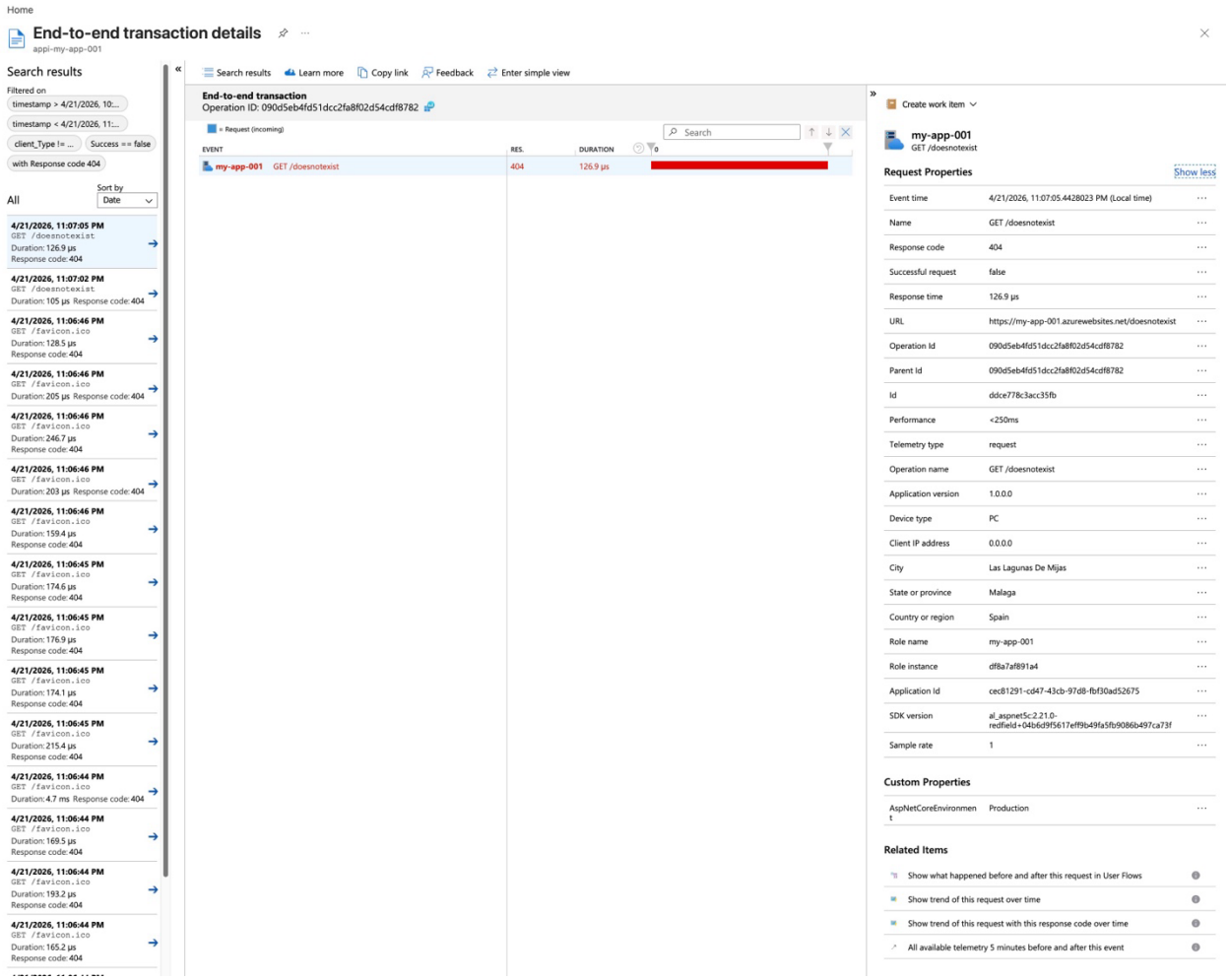
1. Once the web app is back up, hit it a few times: open `https://my-app-001.azurewebsites.net` and refresh 5–10 times. Visit a non-existent URL like https://my-app-001.azurewebsites.net/**doesnotexist** to generate 404s.
2. Wait 2–3 minutes for telemetry to flow (there's a small ingestion delay).
3. Go to the Application Insights service `appi-my-app-001` → **Overview**. You should see request and failure counters populating in the live charts.
4. Click **Investigate** → **Live metrics** to watch requests arrive in real time. Hit the site a few more times to see the graph move.

> **TIP**
> Live Metrics is one of the best-kept secrets of Application Insights. It shows you the last 60 seconds of activity with no ingestion delay — perfect for 'is my latest deploy breaking things?' moments. It's also free (no ingestion cost for what you see on that page).




appi-my-app-001 | Failures
Application Insights
Search
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Resource visualizer
Investigate
Application map
Smart detection
Live metrics
Search
Availability
Failures
Performance
Agents (preview)
Monitoring
Alerts
Metrics
Diagnostic settings
Logs
Workbooks
Dashboards with Grafana
Usage
Configure
Settings
Automation
Help
Refresh
View in Logs
Analyze with Workbooks
Copy link
Feedback
Server
Browser
Local Time: Last hour
Roles = All
Operations
Dependencies
Exceptions
Roles
Failed request count
Request count
Select operation
Search to filter items...
OPERATION NAME
COUNT (FAILED)
COUNT
Overall
0
0
Overall
Top 3 response co...
COUNT
FILTER...
404
18
click here
Top 3 exception ty...
Top 3 failed depen...
Select a sample operation
Filtered on client_Type !...
Success == false
with Response code 404
All
Sort by
Date
4/21/2026, 11:07:05 PM
GET /doesnotexist
Duration: 126.9 µs Response code: 404
4/21/2026, 11:07:02 PM
GET /doesnotexist
Duration: 105 µs Response code: 404
4/21/2026, 11:06:46 PM
GET /favicon.ico
Duration: 128.5 µs Response code: 404
4/21/2026, 11:06:46 PM
GET /favicon.ico
Duration: 205 µs Response code: 404
4/21/2026, 11:06:46 PM
GET /favicon.ico
Duration: 246.7 µs Response code: 404
4/21/2026, 11:06:46 PM
GET /favicon.ico
Duration: 203 µs Response code: 404
4/21/2026, 11:06:46 PM
GET /favicon.ico
Duration: 159.4 µs Response code: 404
4/21/2026, 11:06:45 PM
GET /favicon.ico
Duration: 174.6 µs Response code: 404
4/21/2026, 11:06:45 PM
GET /favicon.ico
Duration: 176.9 µs Response code: 404
4/21/2026, 11:06:45 PM
GET /favicon.ico
Duration: 174.1 µs Response code: 404
4/21/2026, 11:06:45 PM
GET /favicon.ico
Duration: 215.4 µs Response code: 404
4/21/2026, 11:06:44 PM
GET /favicon.ico
Duration: 4.7 ms Response code: 404
4/21/2026, 11:06:44 PM
GET /favicon.ico
Duration: 169.5 µs Response code: 404


Home
End-to-end transaction details
appi-my-app-001
Search results
Search results
Learn more
Copy link
Feedback
Enter simple view
End-to-end transaction
Operation ID: 090d5eb4fd51dcc2fa8f02d54cdf8782
Request (incoming)
Search
EVENT
RES.
DURATION
my-app-001 GET /doesnotexist
404
126.9 µs
Create work item
my-app-001
GET /doesnotexist
Request Properties
Show all
Event time
4/21/2026, 11:07:05.4428023 PM (Local time)
Name
GET /doesnotexist
Response code
404
Successful request
false
Response time
126.9 µs
URL
https://my-app-001.azurewebsites.net/doesnotexist
Custom Properties
AspNetCoreEnvironment
Production
Related Items
Show what happened before and after this request in User Flows
Show trend of this request over time
Show trend of this request with this response code over time
All available telemetry 5 minutes before and after this event


Home
End-to-end transaction details
appi-my-app-001
Search results
Filtered on
timestamp > 4/21/2026, 10:...
timestamp < 4/21/2026, 11:...
client_Type != ...
Success == false
with Response code 404
All
Sort by
Date
4/21/2026, 11:07:05 PM
GET /doesnotexist
Duration: 126.9 µs
Response code: 404
4/21/2026, 11:07:02 PM
GET /doesnotexist
Duration: 105 µs Response code: 404
4/21/2026, 11:06:46 PM
GET /favicon.ico
Duration: 128.5 µs
Response code: 404
4/21/2026, 11:06:46 PM
GET /favicon.ico
Duration: 205 µs Response code: 404
4/21/2026, 11:06:46 PM
GET /favicon.ico
Duration: 246.7 µs
Response code: 404
4/21/2026, 11:06:46 PM
GET /favicon.ico
Duration: 203 µs Response code: 404
4/21/2026, 11:06:46 PM
GET /favicon.ico
Duration: 159.4 µs
Response code: 404
4/21/2026, 11:06:45 PM
GET /favicon.ico
Duration: 174.6 µs
Response code: 404
4/21/2026, 11:06:45 PM
GET /favicon.ico
Duration: 176.9 µs
Response code: 404
4/21/2026, 11:06:45 PM
GET /favicon.ico
Duration: 174.1 µs
Response code: 404
4/21/2026, 11:06:45 PM
GET /favicon.ico
Duration: 215.4 µs
Response code: 404
4/21/2026, 11:06:44 PM
GET /favicon.ico
Duration: 4.7 ms Response code: 404
4/21/2026, 11:06:44 PM
GET /favicon.ico
Duration: 169.5 µs
Response code: 404
4/21/2026, 11:06:44 PM
GET /favicon.ico
Duration: 193.2 µs
Response code: 404
4/21/2026, 11:06:44 PM
GET /favicon.ico
Duration: 165.2 µs
Response code: 404
Search results
Learn more
Copy link
Feedback
Enter simple view
End-to-end transaction
Operation ID: 090d5eb4fd51dcc2fa8f02d54cdf8782
Request (incoming)
Search
EVENT
RES.
DURATION
my-app-001 GET /doesnotexist
404
126.9 µs
Create work item
my-app-001
GET /doesnotexist
Request Properties
Show less
Event time 4/21/2026, 11:07:05.4428023 PM (Local time)
Name GET /doesnotexist
Response code 404
Successful request false
Response time 126.9 µs
URL https://my-app-001.azurewebsites.net/doesnotexist
Operation Id 090d5eb4fd51dcc2fa8f02d54cdf8782
Parent Id 090d5eb4fd51dcc2fa8f02d54cdf8782
Id ddce778c3acc35fb
Performance <250ms
Telemetry type request
Operation name GET /doesnotexist
Application version 1.0.0.0
Device type PC
Client IP address 0.0.0.0
City Las Lagunas De Mijas
State or province Malaga
Country or region Spain
Role name my-app-001
Role instance df8a7af891a4
Application Id cec81291-cd47-43cb-97d8-fbf30ad52675
SDK version
Sample rate 1
Custom Properties
AspNetCoreEnvironment Production
Related Items
Show what happened before and after this request in User Flows
Show trend of this request over time
Show trend of this request with this response code over time
All available telemetry 5 minutes before and after this event

7.4 Confirm the App Settings that were added

Turning on App Insights added a set of App Settings to your Web App. Knowing which ones exist is useful because in real work you'll often set these via Bicep, Terraform, or a CI pipeline rather than the portal.

1. Navigate to `my-app-001` → **Settings** → **Environment variables**.
2. Look for these new App Settings (the auto-instrumentation added them):
   - `APPLICATIONINSIGHTS_CONNECTION_STRING` — the modern endpoint for App Insights ingestion
   - `ApplicationInsightsAgent_EXTENSION_VERSION` = `~3` — tells the runtime to load the agent
   - `XDT_MicrosoftApplicationInsights_Mode` = `recommended` — the collection profile
3. Do NOT remove these.

   PS: Your `SqlConnectionString` from Lab 2 should also still be there — confirm it is.

8. Enable Diagnostic Settings on All Three Services

Diagnostic Settings route a resource's logs (and optionally its metrics) to a destination like Log Analytics. Without them, the resource still emits telemetry internally but you can't query it with KQL and you can't write log-based alerts on it.

> 📘 **NOTE**
> You only need to configure diagnostic settings once per resource. Each resource can have up to 5 diagnostic settings (useful when you want to send the same data to both a workspace AND a Storage Account for long-term archive, for example).

8.1 Web App (my-app-001)

1. Navigate to the Web App `my-app-001`.
2. In the left menu, click **Monitoring** → **Diagnostic settings**.
3. Click **+ Add diagnostic setting**.
4. Configure:
   - **Diagnostic setting name:** `diag-to-log-analytics`
   - **Logs — check these categories:** HTTP Logs, App Service Console Logs, App Service App Logs, App Service Audit Logs, IPSec Audit Logs, App Service Platform Logs, App Service Authentication logs (preview)
   - **Metrics:** check **AllMetrics**
   - **Destination:** check **Send to Log Analytics workspace**
   - **Subscription:** your subscription
   - **Log Analytics workspace:** `log-monitoring-001 (westeurope)`
5. Click **Save**.

What each log category gives you

Category	What it contains
AppServiceHTTPLogs	Every HTTP request: URL, method, status, response time, client IP. This is the main one you'll query.
AppServiceConsoleLogs	stdout/stderr from your app. If your code calls Console.WriteLine, it ends up here.
AppServiceAppLogs	Application-level log messages written through ASP.NET logging (ILogger) at or above the configured level.
AppServiceAuditLogs	Who deployed what and when (Kudu deployment audit).
AppServiceIPSecAuditLogs	IP-based access restriction events (who was allowed/denied).
AppServicePlatformLogs	Azure platform events for the app instance (restarts, scale operations, etc.).

> ⚠ **WARNING**
> Each enabled log category ingests data that is billed by volume (roughly **€2.50 per GB ingested** for Pay-as-you-go Log Analytics). For production, be selective: enable the categories you actually use. **HTTP Logs** and **AppLogs** cover 90% of real debugging needs. **IPSecAuditLogs** on a quiet app produce almost nothing, so they're free anyway — but a chatty app with DEBUG-level AppLogs can burn through your free tier in hours.

8.2 SQL Database (dbwebapp)

1. Navigate to the SQL Database `dbwebapp` (not the server).
2. In the left menu, click **Monitoring** → **Diagnostic settings**.
3. Click **+ Add diagnostic setting**.
4. Configure:
   - **Diagnostic setting name:** `diag-to-log-analytics`
   - **Logs — check:** SQLInsights, AutomaticTuning, QueryStoreRuntimeStatistics, QueryStoreWaitStatistics, Errors, DatabaseWaitStatistics, Timeouts, Blocks, Deadlocks, DevOpsOperationsAudit, SQLSecurityAuditEvents
   - **Metrics:** check **Basic**, **InstanceAndAppAdvanced**, **WorkloadManagement**
   - **Destination:** check **Send to Log Analytics workspace** → **log-monitoring-001**
5. Click **Save**.

Critical SQL diagnostic categories to know

Category	Why it matters
Errors	Every SQL error (login failed, invalid object, etc.). Your first stop when the app suddenly can't query.
Timeouts	Query timeouts. Correlate with CPU / waits to find slow queries.
Blocks	Queries blocked by other queries. Spike here usually means a long-running transaction is holding locks.
Deadlocks	Two queries waiting for each other. SQL kills one — but you want to know which.
QueryStoreRuntimeStatistics	Per-query performance history. Great for "which query suddenly got slow?".

8.3 Storage Account (sawebappprod001)

Storage Account diagnostic settings are slightly different — there isn't one single Diagnostic Settings blade. Instead, each service inside the storage account (Blob, File, Queue, Table) has its own. For this lab we'll enable Blob-level diagnostics because that's what our app uses.

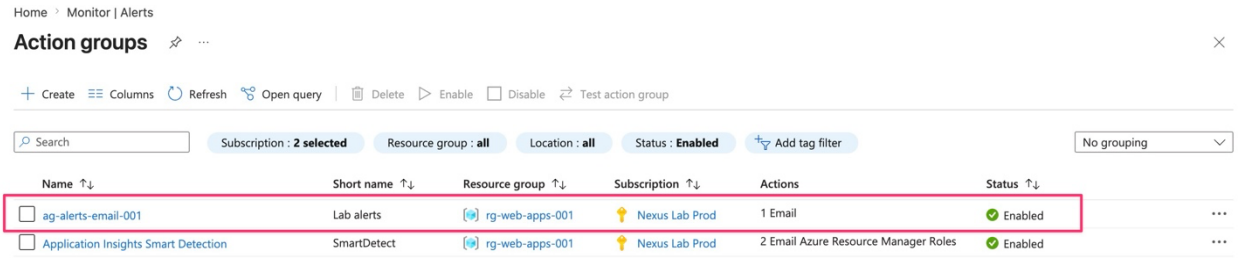
1. Navigate to the Storage Account `sawebappprod001`.
2. In the left menu, scroll down and click **Monitoring** → **Diagnostic settings**.
3. You'll see a list of sub-resources: `sawebappprod001`, **`blobServices`**`/default`, `fileServices/default`, `queueServices/default`, `tableServices/default`.
4. Click **+ Add diagnostic setting** next to the `blobServices/default` row.
5. Configure:
    - **Diagnostic setting name:** `diag-blob-to-log-analytics`
    - **Logs — check:** StorageRead, StorageWrite, StorageDelete
    - **Metrics:** check **Transaction**
    - **Destination:** check **Send to Log Analytics workspace** → **log-monitoring-001**
6. Click **Save**.
7. (Optional) Repeat for the parent `sawebappprod001` row if you want account-level metrics. For this lab, blob-only is enough.

> 🚧 **COMMON PITFALL**
> It's easy to land on the parent storage account's Diagnostic settings page and see nothing actionable. Make sure you pick the right **sub-resource** row (`blobServices/default`). That's where Blob-level logs live.

9. Create an Action Group for Email Notifications

An Action Group is a reusable collection of notification targets. Every alert rule references an action group to know 'when I fire, who to tell'. Centralizing this means you can update the email list in one place instead of editing 20 alerts.

1. In the Portal, search for **Monitor** and open the Monitor service.
2. In the left menu, click **Alerts** → **Action groups**.
3. Click **+ Create**.
4. On the Basics tab:
    - **Subscription:** your subscription
    - **Resource group:** `rg-web-apps-001`
    - **Region: Global** (Action groups are global — leave as Global)
    - **Action group name:** `ag-alerts-email-001`
    - **Display name:** `Lab alerts` (short; this goes into every alert email subject — max 12 chars)
5. On the Notifications tab, add one notification:
    - **Notification type: Email/SMS message/Push/Voice**
    - **Name:** `email-me`
    - In the pane that opens: check **Email** and enter your email address.
    - Click **OK** to save the notification.
6. On the **Actions** tab, leave empty (Actions are things like Logic Apps, Runbooks, Functions — overkill for this lab).
7. Click **Review + create**, then **Create**.

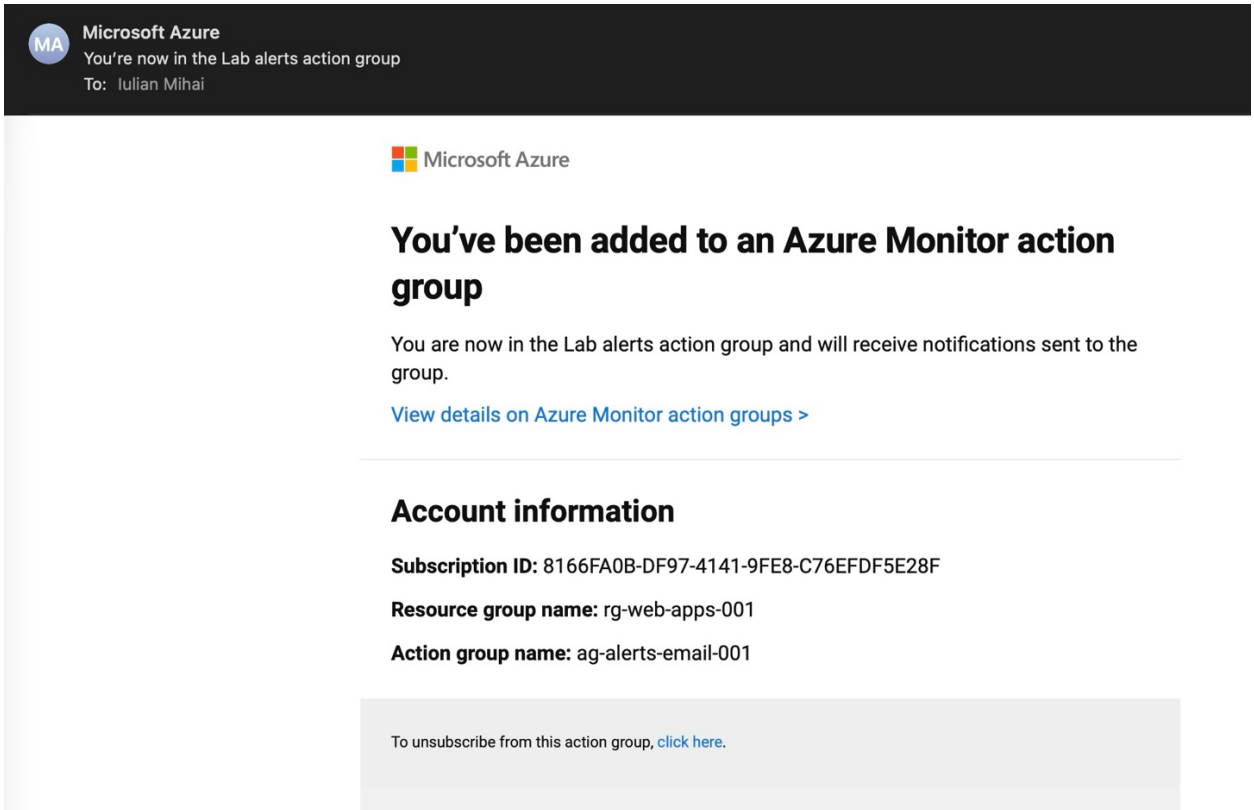


9.1 Confirm you received the test email

Azure sends a one-time test email when you add a new email notification, so it should already be in your inbox.

1. Check your inbox for a message from azure-noreply@microsoft.com with subject starting with "Microsoft Azure: You have been added..."
2. If you don't see it, check your spam folder. Mark the address as 'not spam' to ensure real alerts don't also go there.
3. If it's truly missing, go back to the action group → click the email entry → Edit → click 'Send test' at the bottom.

> ⚠ **WARNING**
>
> Azure enforces rate limits on email notifications to prevent abuse: a single email recipient can receive at most 100 emails per hour from a given action group. In a flood scenario, additional alerts are dropped. In production, route heavy traffic through ITSM tools like PagerDuty or ServiceNow, not raw email.



10. Create Metric Alerts

Metric alerts are evaluated against Azure's platform metrics (CPU, memory, request count, etc.) and can fire within 1–5 minutes of the condition becoming true. You'll create several across the three services.

All metric alerts follow the same flow in the portal:

- **Scope** — which resource(s) the alert watches
- **Condition** — the signal (the specific metric), the aggregation (avg, max, count...), the threshold, and the evaluation frequency
- **Actions** — the Action Group(s) to call when the condition is met
- **Details** — name, severity, description

10.1 Web App: HTTP 5xx and Response Time

Alert 1 — HTTP 5xx errors

1. In the Portal, search for **Monitor**, then go to **Alerts** → **+ Create** → **Alert rule**.
2. On the Scope tab, click Select scope. Filter to the resource group rg-web-apps-001, choose resource type App Services, and pick my-app-001. Click Apply.
3. On the Condition tab, click See all signals, then select Http Server Errors.
4. Configure the condition:
    - **Threshold: Static**
    - **Aggregation type: Total**
    - **Operator: Greater than or equal to**
    - **Threshold value:** `5`
    - **Unit:** (count)
    - **Aggregation granularity:** `5 minutes`
    - **Frequency of evaluation:** `1 minute`
5. On the **Actions** tab, click **+ Select action groups** and choose `ag-alerts-email-001`.
6. On the Details tab:
    - **Severity: 2 - Warning**
    - **Alert rule name:** `al-webapp-5xx-001`
    - **Alert rule description:** `Fires when the web app returns 5 or more HTTP 5xx responses in a 5-minute window.`
    - **Region:** West Europe
    - **Enable upon creation: Yes**
    - **Automatically resolve alerts: Yes**
7. Click **Review + create**, then **Create**.

Alert 2 — Response Time

Create a second alert the same way — the only differences are the signal and threshold.

1. Same scope (`my-app-001`), same action group.
2. Signal: **Response Time**
3. Condition:
    - **Aggregation: Average**
    - **Operator: Greater than, Threshold:** `3` (seconds)
    - **Granularity:** `5 minutes`, **Frequency:** `1 minute`
4. **Severity: 3 - Informational** (slow is not broken)
5. **Name:** `al-webapp-slow-response-001`

> 💡 **TIP**
> Why is 'slow' sev 3 but 5xx sev 2? Because 5xx means users are seeing error pages — that's a functional regression. Slow means the site works but feels sluggish — that's a performance regression. In triage, functional issues outrank performance ones. This is a judgment call and teams tune it to their own taste.

10.2 Web App: CPU and Memory

F1 (Free) App Service plan metrics are limited — CPU and memory percentages don't surface on Free/Shared tiers (you saw the "metrics are not applicable to Free and Shared SKU" banner in Lab 1). The standard alternative is to use the counters from the App Service Plan.

Alert 3 — CPU time on the Plan

1. Create a new alert rule.
2. Scope: `asp-my-app-001` (the App Service **plan**, not the web app).
3. Signal: **CPU Time**
4. Condition: Aggregation Total, Operator Greater than, Threshold 60, Granularity 15 minutes, Frequency 5 minutes.
5. **Name:** `al-asp-cpu-high-001` · **Severity: 2 - Warning**

> **NOTE**
> F1 is capped at 60 CPU minutes per day. This alert fires when the app has used most of its daily budget in a single 15-minute window — an early warning that you'll hit the throttle soon. On a paid SKU (B1+), you'd use 'CPU Percentage' with a threshold around 80%.

Alert 4 — Memory Working Set

1. Create a new alert rule.
2. Scope: `my-app-001` (the web app itself this time).
3. Signal: **Memory working set**
4. Condition: Aggregation Average, Operator Greater than, Threshold 800000000 (800 MB), Granularity 5 minutes, Frequency 1 minute.
5. **Name:** `al-webapp-memory-high-001` · **Severity: 2 - Warning**

10.3 SQL Database: vCore, Failed Connections, Deadlocks

Alert 5 — CPU/vCore percent

1. Create a new alert rule.
2. Scope: `dbwebapp` (the database, not the server).
3. Signal: **CPU percentage**
4. Condition: Aggregation Average, Greater than 80, Granularity 5 min, Frequency 1 min.
5. **Name:** `al-sqldb-cpu-80-001` · **Severity: 2 - Warning**

Alert 6 — Failed Connections

1. Create a new alert rule.
2. Scope: `dbwebapp`.
3. Signal: **Failed Connections**
4. Condition: Aggregation Total, Greater than or equal to 5, Granularity 5 min, Frequency 1 min.
5. **Name:** `al-sqldb-failed-conn-001` · **Severity: 2 - Warning**

Alert 7 — Deadlocks

1. Create a new alert rule.
2. Scope: `dbwebapp`.
3. Signal: **Deadlocks**
4. Condition: Aggregation Total, Greater than or equal to 1, Granularity 5 min, Frequency 1 min.
5. **Name:** `al-sqldb-deadlocks-001` · **Severity: 1 - Error** (any deadlock is already a bug)

10.4 Storage: Availability and Server Errors

Alert 8 — Availability

1. Create a new alert rule.
2. Scope: `sawebappprod001` → **Resource type: Storage account/default/blob** (pick the blob sub-resource).
3. Signal: **Availability**
4. Condition: Aggregation Average, Less than 99, Granularity 5 min, Frequency 1 min.
5. **Name:** `al-storage-availability-001` · **Severity: 1 - Error**

Alert 9 — Blob 5xx transactions

1. Create a new alert rule.
2. Scope: same blob sub-resource as Alert 8.
3. Signal: **Transactions**
4. Add a **Split by dimension** on `ResponseType` and filter to values starting with `Server` (ServerBusyError, ServerOtherError, ServerTimeoutError).
5. Condition: Aggregation Total, Greater than or equal to 5, Granularity 5 min.
6. **Name:** `al-storage-5xx-001` · **Severity: 2 - Warning**

> **NOTE**
> Splitting by dimension is powerful — the single rule covers all three 'Server*' response types. Without splitting, you'd need three separate rules.

11. Create Log-Based Alerts (KQL)

Log alerts evaluate a KQL query on a schedule and fire when the result meets a condition. They're essential when you need to filter, correlate, or express a condition that isn't a simple metric threshold.

11.1 A gentle KQL primer

If you've never written KQL before, the basics are simple. A query starts with a table name and uses the pipe (|) to chain operators, like piping commands in a shell.

```
// The general shape:
TableName
| where <filter conditions>
| summarize <aggregations> by <grouping>
| order by <columns>
```

Concrete example — count App Insights exceptions per type in the last hour:

```
exceptions
| where timestamp > ago(1h)
| summarize count() by type
| order by count_ desc
```

11.2 Alert 10 — Exceptions in App Insights

This alert watches the App Insights exceptions table and fires when the app is throwing any exceptions at all.

1. Navigate to `appi-my-app-001`.
2. In the left menu, click **Monitoring** → **Logs** (this opens the KQL query editor scoped to App Insights).
3. First, test the query. Paste and run:

```
exceptions
| where timestamp > ago(10m)
| summarize ExceptionCount = count() by bin(timestamp, 5m), type
| order by timestamp desc
```

4. If the app has no exceptions yet, the result will be empty — that's expected.
5. Now click **+ New alert rule** at the top of the logs pane. The portal pre-fills the scope with the App Insights resource and the query with the one in the editor.
6. On the Condition tab, verify the query is present, then configure:
    - **Measurement:** Table rows (count of rows returned)
    - **Aggregation granularity:** `5 minutes`
    - **Operator: Greater than, Threshold:** `0`
    - **Frequency of evaluation:** `5 minutes`
7. Action group: ag-alerts-email-001.
8. Details:
    - **Severity: 1 - Error**
    - **Name:** `al-appi-exceptions-001`

- **Description:** `Fires whenever the app reports any exceptions in App Insights.`

9. Click **Review + create**, then **Create**.

> 🚧 **COMMON PITFALL**
> When creating a log alert from the Logs pane, always use **+ New alert rule** at the top of that pane — not Alerts → + Create. Creating the alert from Alerts forces you to re-select the scope and paste the query manually, which is error-prone. The Logs-pane flow pre-fills everything correctly.

11.3 Alert 11 — Storage anonymous access events (optional bonus)

Since our images container has anonymous access enabled, it's smart to watch for unusual access patterns. This KQL counts anonymous blob read requests per minute and fires if the rate is unusually high (might indicate hotlinking or scraping).

```
StorageBlobLogs
| where OperationName == "GetBlob"
| where AuthenticationType == "Anonymous"
| summarize AnonReads = count() by bin(TimeGenerated, 1m)
| where AnonReads > 100
```

Configure the same way as Alert 10 but scoped to the storage account and with a threshold of Table rows > 0 (i.e., 'there was any 1-minute bucket with > 100 anonymous reads'). For the lab, this is optional — we'll only build Alert 10 in the step-by-step.

12. Create an Activity Log Alert

Activity Log alerts fire on Azure control-plane events — things like 'someone deleted a resource', 'someone changed the firewall rules', 'a service health issue was posted for West Europe'. These are separate from metric and log alerts because they watch the Azure Resource Manager, not telemetry from the resources themselves.

Alert 12 — Resource deleted in the RG

1. Go to **Monitor** → **Alerts** → **+ Create** → **Alert rule**.
2. Scope: select the resource group `rg-web-apps-001` itself (not a specific resource).
3. Condition: click **See all signals**. Switch the Signal type filter to **Activity Log**.
4. Select the signal **Delete Resource** (the generic resource-delete event — listed as `Delete Resource (Microsoft.Resources/subscriptions/resourceGroups)` or similar).
5. On the **Logic** tab, set **Event Level** = **All**, **Status** = **Succeeded**.
6. Action group: ag-alerts-email-001.
7. Details:
    - **Name:** `al-rg-resource-delete-001`
    - **Severity: 2 - Warning** (intentional deletes are normal; unintentional ones are exactly what you want to catch)
8. Click **Create**.

> **NOTE**
> Activity Log alerts are free — Azure doesn't charge for the evaluation because the Activity Log always exists and is read with no ingestion cost. Scale them up freely: one alert per sensitive resource type is a common pattern.

Variants you'd build in real environments

- **Firewall rule change on SQL server** (signal: Update or Delete SQL server firewall rules)
- **Storage account key regenerated** (signal: List Storage Account Keys or Regenerate Key)
- **Role assignment created on the subscription** (signal: Create role assignment — catches privilege escalation)
- **Service Health issue affecting your region** (signal: Service Health → Incident)

13. Build a Monitoring Dashboard

Alerts tell you when something is broken. Dashboards tell you what 'normal' looks like. In this section you'll build a simple shared dashboard with the key metrics from all three services, so you can glance at it in the morning and catch drift before it becomes an alert.

1. In the Portal, click the top-left **hamburger menu** → **Dashboards** (or search 'Dashboard' in the top bar).
2. Click **+ New dashboard** → **Blank dashboard**.
3. Name it `dash-monitoring-001` and click **Save**.

13.1 Pin the Web App request chart

1. Open a second browser tab and navigate to `my-app-001` → **Monitoring** → **Metrics**.
2. Create a new chart:
   - **Metric: Requests** (count, sum)
   - **Add metric**: **Http Server Errors** (count, sum)
   - **Time range:** `Last 4 hours`, 5 minute granularity
3. At the top of the chart, click the **pin icon** → **Pin to dashboard** → select `dash-monitoring-001`.

13.2 Pin the SQL Database metrics

1. Navigate to `dbwebapp` → **Monitoring** → **Metrics**.
2. Create a chart with: CPU percentage, Successful Connections, Failed Connections.
3. Pin to the same dashboard.

13.3 Pin the Storage transactions chart

1. Navigate to `sawebappprod001` → **Monitoring** → **Metrics**.
2. Create a chart with: Transactions (splitting by ResponseType), Ingress, Egress.
3. Pin to the dashboard.

13.4 Pin a live KQL query

1. Navigate to `log-monitoring-001` → **Logs**.
2. Paste and run:

```
AppServiceHTTPLogs
| where TimeGenerated > ago(1h)
| summarize RequestCount = count() by bin(TimeGenerated, 5m), ScStatus
| render columnchart
```

3. Click **Pin to** → **Azure dashboard** → `dash-monitoring-001`.

13.5 Make the dashboard shareable (optional)

1. Return to your dashboard and click Share at the top.
2. Publish to the same resource group so teammates with Reader role on the RG can see it.

14. Break Things On Purpose

Configuring alerts is only half the exercise — you need to see them fire so you trust them. This section walks you through generating the signals that trigger each alert you built. Do each in order; after each one, keep an eye on your email inbox.

> ⚠ **WARNING**
> The SQL database is Serverless with a 1-hour auto-pause delay. If it's paused when you start, the first SQL-related test will take 30–60s to resume the database. This is normal.

14.1 Trigger the 404-heavy load (but not 5xx yet)

This will drive the Request count and Response Time charts even if it doesn't trip an alert.

PowerShell (Windows):

```
1..50 | ForEach-Object {
  try { Invoke-WebRequest -Uri "https://my-app-001.azurewebsites.net/doesnotexist" -
UseBasicParsing | Out-Null } catch {}
  Start-Sleep -Milliseconds 200
}
```

bash (macOS / Linux):

```
for i in $(seq 1 50); do
  curl -sS -o /dev/null "https://my-app-001.azurewebsites.net/doesnotexist"
  sleep 0.2
done
```

14.2 Trigger al-webapp-5xx-001

The simplest way to cause deliberate 5xx errors is to break the SqlConnectionString — the app's exception handler returns HTTP 500 when the DB isn't reachable.

1. Navigate to `my-app-001` → **Settings** → **Environment variables**.
2. Edit `SqlConnectionString` and change the password to `WrongPassword123!`. Click **Apply** → **Apply**.
3. Wait ~30 seconds for the app to restart, then hit `https://my-app-001.azurewebsites.net/` 10–15 times.
4. You will see the red 'Database error: Login failed for user sqldba' page in the browser — this counts as HTTP 500 in the App Service metrics.
5. Wait 5–10 minutes for the metric alert to evaluate and the email to arrive.
6. **Revert the password** immediately after you see the alert fire. You don't want to stay broken longer than needed.

14.3 Trigger al-appi-exceptions-001

Same mechanism — the wrong-password scenario throws SqlException inside the app, which App Insights captures. This alert should fire within ~10 minutes of the first batch of bad requests (since the query is evaluated every 5 minutes).

14.4 Trigger al-sqldb-failed-conn-001

1. While the wrong password is still applied (or reapply it), the SQL server will record Failed Connections each time the app tries to authenticate.
2. Hit the Web App 10 times in a minute via any method (browser refresh is fine).
3. Each attempt produces one Failed Connection metric — this will cross the threshold of 5 in a 5-minute window and the alert will fire within a few minutes.

14.5 Trigger al-storage-availability-001 (harder to simulate)

Availability is a platform metric set by Microsoft's own probes against Azure Storage — you can't easily make it drop as a tenant. In the real world, you would see this fire during genuine regional incidents. For the lab, there are two safe ways to exercise it:

- **Option A — Just trust it.** Leave the rule configured. Your resource will accumulate history that you'll see rise and fall over time in Metrics.
- **Option B — Break the container permissions temporarily.** Change the `images` container's anonymous access level from **Blob** to **Private**. Hit the Web App: the `<img>` tag will now 404 (authorization failed). Wait 5 min, confirm you see server-error transactions in Metrics, then revert to Blob.

14.6 Trigger al-rg-resource-delete-001

1. Navigate to `sawebappprod001` → **Data storage** → **Containers**.
2. Click **+ Add container**, create a throwaway container named `throwaway-delete-me` (Private access level).
3. Delete that container.
4. Within 2–5 minutes you'll receive an email: "Activity Log Alert: al-rg-resource-delete-001 fired."

> 🚧 **COMMON PITFALL**
> Activity Log alerts have the longest evaluation delay — typically 2–5 minutes, sometimes up to 10. Don't assume the alert isn't working just because the email didn't arrive in 30 seconds. Check **Monitor** → **Alerts** → **Alert history** to see whether the rule fired even if the email hasn't arrived yet.

15. Verify All Alerts Fired

Time to check your work. Every alert you configured should have a record — either of firing (triggered by Section 14) or of being evaluated but not crossing the threshold.

1. Go to **Monitor** → **Alerts**. The overview shows a summary of the last 24 hours: Total alerts, Smart groups, Action groups.
2. Click **Alerts** in the left nav to see the list of fired alerts. You should see entries from Section 14.
3. For each alert you triggered, click it and review:
    - **Condition** — what the alert fired on
    - **Signal** — the actual metric value at the time of firing
    - **Timeline** — when it fired, when it resolved
    - **Actions taken** — should say "Action group 'ag-alerts-email-001' succeeded"
4. Open one of the Azure alert emails in your inbox. Confirm it contains: alert rule name, affected resource, signal value, a deep link back to the alert in the portal, and the severity.

Expected alert inventory

Alert rule	Fires on	Should have fired in Section 14?
al-webapp-5xx-001	5+ HTTP 5xx in 5 min	Yes (14.2)
al-webapp-slow-response-001	avg response time > 3s	Maybe (cold start)
al-asp-cpu-high-001	Plan CPU time > 60 in 15 min	Unlikely in lab
al-webapp-memory-high-001	Memory > 800 MB	Unlikely in lab
al-sqldb-cpu-80-001	DB CPU > 80%	Unlikely in lab
al-sqldb-failed-conn-001	5+ failed conns in 5 min	Yes (14.4)
al-sqldb-deadlocks-001	any deadlock	Unlikely in lab
al-storage-availability-001	availability < 99%	No (platform-level)
al-storage-5xx-001	5+ server-error blob transactions	Maybe (14.5 optional)
al-appi-exceptions-001	any exception in App Insights	Yes (14.3)
al-rg-resource-delete-001	any delete resource action	Yes (14.6)

> 💡 **TIP**
> Not every alert fires in the lab, and that's fine — the point is to demonstrate the plumbing. In production, alerts often go weeks without firing. That's good. You know they work because you triggered them at least once during test and verified the email arrived.

16. Lab vs. Production

This lab gets you the mechanics, but a production monitoring setup for the same three resources would look different in several important ways.

Area	This Lab	Production
Notification channels	Single email	Email + PagerDuty / OpsGenie / ServiceNow + Teams channel for low-sev
Action groups	One, reused everywhere	One per severity (sev0→page, sev2→Teams, sev4→ticket-only)
Alert scope	Individual resources	Resource groups or subscriptions with dynamic targeting
Thresholds	Static numbers chosen by hand	Dynamic Thresholds (ML-based) for seasonality; or SLO-driven budgets
Retention	30 days (default)	90 days interactive + archive tier for security/compliance
Log categories	Everything enabled	Selective — enable what you query, to manage ingestion cost
App Insights sampling	100% (lab default)	Adaptive sampling to cap cost (target ~€X/month)
Dashboards	One shared lab dashboard	Per-team Workbooks, Grafana, or Managed Grafana instances
Alert documentation	Description field	Every alert links to a runbook (wiki, Confluence) with steps to investigate and remediate
On-call rotation	Just your inbox	PagerDuty/OpsGenie schedule, escalation policy, paging rules by severity
SLIs/SLOs	Implicit (5xx, response time)	Explicit availability and latency SLOs; error budgets tracked in Grafana
Deployment	Portal click-through	Bicep/Terraform — every alert in source control, reviewed in PRs
Change alerting	Activity log delete only	Full config change audit: any diagnostic setting removed, any alert disabled, any firewall opened

Principles worth internalizing

- **Alerts are code.** Keep them in source control (Bicep or Terraform), review changes in PRs, and deploy through pipelines. Clicking in the portal is fine for learning but doesn't scale.
- **Every alert needs a runbook.** When an alert fires at 3 AM, the on-call engineer has 60 seconds to decide: 'is this real, and what do I do?' A 'see runbook link' in the alert description answers that.
- **Measure what users feel.** Availability and end-to-end latency are what users feel. CPU and memory are internal signals. Prefer user-facing signals for paging; use internal ones for dashboards.
- **Alert fatigue is real.** An alerting system with 500 alerts that fire daily trains the team to ignore them. Fewer, higher-signal alerts always beat more, noisier ones.
- **Cost grows with ingestion, not rules.** Log Analytics bills per GB ingested. Enabling a verbose log category on a chatty service is where budgets evaporate — not in the number of alert rules.

17. Cleanup

If you're done with all three labs, delete the entire resource group — same as the previous labs. That's the cleanest option.

1. Portal → **Resource groups** → `rg-web-apps-001`.
2. Click **Delete resource group** at the top.
3. Type `rg-web-apps-001` to confirm, then **Delete**.

If you want to keep the infra but remove only the monitoring

Delete in reverse order of creation:

- All 12 alert rules (Monitor → Alerts → Alert rules → select → Delete)
- The Action Group ag-alerts-email-001
- The Application Insights resource appi-my-app-001 (also removes the App Settings from the Web App? NO — you'll need to remove APPLICATIONINSIGHTS_CONNECTION_STRING and related settings manually)
- The Log Analytics workspace log-monitoring-001
- The Dashboard dash-monitoring-001

> ⚠ **WARNING**
> Deleting the Log Analytics workspace does NOT immediately delete the ingested data — Azure keeps it for a soft-delete recovery window (default 14 days). If you want data permanently erased for privacy reasons, there's a Purge API, but for this lab the default is fine.

Appendix A — KQL Query Library

A starter library of useful KQL queries for your workspace. Paste any of these into Log Analytics (log-monitoring-001 → Logs) to run them.

A.1 Top 10 slowest requests in the last hour

```
requests
| where timestamp > ago(1h)
| top 10 by duration desc
| project timestamp, name, resultCode, duration, operation_Id
```

A.2 HTTP status code distribution (Web App)

```
AppServiceHTTPLogs
| where TimeGenerated > ago(24h)
| summarize Count = count() by ScStatus
| order by Count desc
```

A.3 Failed requests grouped by URL

```
requests
| where timestamp > ago(1h)
| where success == false
| summarize FailCount = count() by name, resultCode
| order by FailCount desc
```

A.4 Exception trend

```
exceptions
| where timestamp > ago(24h)
| summarize Count = count() by bin(timestamp, 1h), type
| render timechart
```

A.5 SQL errors in the last hour

```
AzureDiagnostics
| where ResourceProvider == "MICROSOFT.SQL"
| where Category == "Errors"
| where TimeGenerated > ago(1h)
| project TimeGenerated, error_number_d, error_state_d, Message
| order by TimeGenerated desc
```

A.6 Blob transactions by operation type

```
StorageBlobLogs
| where TimeGenerated > ago(1h)
| summarize Count = count() by OperationName, StatusCode
| order by Count desc
```

A.7 Anonymous blob reads (potential hotlinking)

```
StorageBlobLogs
| where OperationName == "GetBlob"
| where AuthenticationType == "Anonymous"
| summarize Reads = count() by bin(TimeGenerated, 5m), CallerIpAddress
| order by TimeGenerated desc
```

A.8 Activity Log — resource delete events

```
AzureActivity
| where OperationNameValue endswith "/delete"
| where ActivityStatusValue == "Success"
| where TimeGenerated > ago(7d)
| project TimeGenerated, Caller, ResourceId, OperationNameValue
| order by TimeGenerated desc
```

Appendix B — Alert Severity Guide

Azure supports five severity levels. There's no universal 'correct' mapping — every team picks its own — but the following is a good starting template.

Severity	Meaning	Typical response	Example
Sev 0 — Critical	Production is down for users	Page on-call 24/7	Website returns 5xx for > 50% of requests for 2 min
Sev 1 — Error	Significant degradation or data issue	Page during business hours, email off-hours	Deadlocks occurring; storage server errors spiking
Sev 2 — Warning	Problem approaching; not yet user-visible	Email, handle next business day	CPU > 80% for 10 min; memory nearing cap
Sev 3 — Informational	Nice-to-know; no action required	Email / ticket	Response time trending up; slow SQL query detected
Sev 4 — Verbose	Audit/change tracking	Logged for audit, no notification	Someone modified a resource group

> **NOTE**
> The ratios matter. In a healthy service, you should have many Sev 3/4 and very few Sev 0/1 rules — and even fewer Sev 0 events actually firing. A team firing Sev 0 alerts weekly is either under-tuning (everything looks critical) or genuinely struggling (real outages). Either way, it's a signal for the team to stop and recalibrate.

Appendix C — Cost Considerations

Approximate costs for the monitoring components of this lab, in West Europe. All prices are indicative — always check the Azure Pricing Calculator for current rates in your region.

Component	Free tier	Beyond free
Log Analytics ingestion	5 GB / month per billing account	~€2.50 / GB ingested
Log Analytics retention	31 days interactive included	~€0.10 / GB / month (interactive beyond 31 days); archive cheaper
Application Insights	(shares the LA workspace allowance)	Same as Log Analytics ingestion
Metric alerts	No ingestion cost	~€0.10 / rule / month, plus €0.10 per 1000 dimension time series
Log (KQL) alerts	No free tier	~€0.50 / rule / month, plus query execution cost
Activity Log alerts	Unlimited, free	€0
Email notifications	1000 / action group / month included	~€1 per 100,000 after
Dashboards	Unlimited, free	€0

Lab cost estimate

If completed and cleaned up in a single day, the monitoring portion of this lab typically ingests well under 100 MB and produces no significant bill — you'll stay inside the 5 GB free tier easily. The alert rules themselves (12 of them) cost roughly €3/month if left running, prorated to about €0.10/day.

> 💡 **TIP**
> The single biggest cost lever in any real monitoring setup is App Insights sampling. By default, App Insights captures 100% of traces. For a high-traffic app that's ruinously expensive. Set adaptive sampling with a target (e.g. 5 requests/sec) and you'll keep cost predictable while retaining statistical fidelity. This is configured in the App Insights resource → Usage and estimated costs → Data volume cap.

Congratulations 🎉

You've taken a web application from 'deployed but blind' to 'deployed, observed, and alerted'. Concretely you can now:

- Route diagnostic logs from Azure resources into a central Log Analytics workspace
- Instrument a .NET Web App with Application Insights without touching the code
- Design metric alerts with appropriate thresholds and severities
- Write basic KQL queries to define log-based alerts
- Audit changes in your environment with Activity Log alerts
- Build a dashboard that gives at-a-glance status for multiple services
- Deliberately trigger alerts to validate that your monitoring actually works

Recommended next modules after Lab 3:

- **Lab 4: Infrastructure as Code.** Rebuild Labs 1, 2, and 3 using Bicep or Terraform, deployed from GitHub Actions. Everything you clicked gets a file in Git.
- **Lab 5: Managed Identity and Key Vault.** Replace the SQL password with a Managed Identity, store any remaining secrets in Key Vault.
- **Lab 6: Private networking.** Add Private Endpoints to SQL and Storage, VNet integration to the Web App, and remove public access to the data services.
- **Lab 7: CI/CD pipelines and deployment slots.** Build a full pipeline with staging slots, smoke tests, and blue/green deploys.

The skill progression is: you built it, you monitored it, next you automate it, then you secure it, then you productionize its delivery. That's the Azure Fundamentals → AZ-104 path.