Admin console logging strategy

Prepared by: Durga Anuhya Yalamarthi

Date: 08/01/2026

Table of Contents

Admin console logging strategy

Objective

To deploy a React/Python application into Azure Container Apps (ACA) while ensuring every administrative action is captured in an immutable (read-only) audit trail within a Log Analytics Workspace.

1. Backend Strategy (Python)

Requirement: Capture every Admin action in a structured, tamper-proof format.

- **The library:** python-json-logger
    - **Why:** Standard Python logs are plain text. Azure needs **JSON** to treat data (like Target_Tenant) as searchable columns rather than just a long string of words.
- **The Command:** logger.info (message, extra={...})
    - **Why:** The extra parameter allows developers to attach the specific audit fields (Admin_User, Action, etc.) directly to the log object.
- **The Transport:** logging.StreamHandler(sys.stdout)

    What is the transport logger?
    Python: The "Transport" is the **StreamHandler**. In your code, when you define logging.StreamHandler(sys.stdout), you are telling the Python logging engine: *"Do not save this to a file on the hard drive; transport it immediately to the* **system's** *active output stream."*

    React: The "Transport" is the **HTTPS protocol** used by the Application Insights SDK. It packages the log and "transports" it over the internet directly to Azure's ingestion servers.

    - **Why:** Azure Container Apps "listens" to the console (STDOUT). By printing logs to the console instead of a file, we ensure the **Azure Logging Agent** captures them instantly.

2. Frontend Strategy (React)

Requirement: Track user actions that happen in the browser (e.g., clicking "Delete User").

- **The library:** @microsoft/applicationinsights-web
    - **Why:** console.log in React only stays in the user's browser. This SDK is the "bridge" that pushes browser events directly into the Azure cloud.

- **The Command:** appInsights.trackEvent()
    - **Why:** This method is specifically designed for custom business logic (like audit trails). It ensures the "Action" and "Target_User" are sent as custom
    - dimensions that match your backend logs.

3. Deployment Strategy (Azure Container Apps)

Requirement: Host the application in a scalable environment with automatic log routing.

- **The Environment:** Azure Container Apps (ACA)
    - **Why:** ACA provides a serverless Kubernetes experience. It includes a "Magic Link" where any console output from the container is automatically routed to the Log Analytics Workspace defined at the environment level.
- **The Storage:** Log Analytics Workspace
    - **Why:** It acts as a **Read-Only Vault**. Once the log is received, it cannot be modified or deleted by the application or the developers, satisfying the
    - **Immutability Rule**.

4. Data Format & Audit Query

Required Format: [Time] [Admin_User] [Action] [Target_Tenant] [Target_User]

To view these logs, use the following **KQL Query** in the Azure Portal:

Code snippet

```
ContainerAppConsoleLogs_CL
| extend d = parse_json(Log_s)
| project
  Time = TimeGenerated,
  Admin = tostring(d.Admin_User),
  Action = tostring(d.Action),
  Tenant = tostring(d.Target_Tenant),
  Target = tostring(d.Target_User)
```

```
| where isnotempty(Admin)

| order by Time desc
```

5. Summary Checklist for Developers

1. **Security Framework:** Use a library (like FastAPI Security) to handle logins; do not write custom auth logic.
2. **JSON Format:** Configure the Python logger to output JSON to sys.stdout.
3. **App Insights:** Initialize the React SDK with the Azure Connection String.
4. **Field Consistency:** Ensure the keys (e.g., Admin_User) are spelled identically in both React and Python code.