🔧 Backend Engineering Task – Pre-signed URL & Microservice Architecture

🎯 Objective

Design and implement **two Dockerized microservices** that collaborate to enable a **secure, traceable file upload workflow** using **pre-signed URLs**. This simulates a scenario where sellers upload product images via a secure and scalable system.

📖 Scenario

A seller is creating a product and wants to upload a product image. The workflow is:

1. Request Upload URL

- The **App Service** exposes an authenticated endpoint.
- The seller sends metadata.
- The App Service signs this metadata and requests a **pre-signed upload URL** from the **Storage Service**.
- The signed URL is returned to the seller.

2. Upload Image

- The seller uploads the image using the pre-signed URL.
- The **Storage Service**:
    - Verifies the **signature**
    - Ensures the uploaded file matches the original metadata
    - Stores the image locally or in-memory
    - Returns an **Image ID**

3. Create Product

- The seller makes a `POST /product` request to the **App Service**, providing product info and the uploaded **Image ID**.
- App Service calls the **Storage Service** to validate that the image ID exists.
- If valid, it saves the product (in-memory is fine) and returns success.

🧠 Evaluation Focus

This task evaluates:

- **Microservice Architecture** & secure inter-service communication
- **Pre-signed URL mechanism**
- **Message integrity checks**
- **Clean, modular code and Dockerization**
- **Basic observability/logging**
- (Optional) Use of **background processing** to demonstrate async understanding

---

🛠️ Technical Requirements

- Language/Stack: **.NET Core (8 or later)** for both services
- API Protocol: REST (with JSON)
- Auth: Basic API key or JWT-based authentication
- Signature: Any secure method
- File Storage: Use **local disk or in-memory** (no need for actual cloud storage)

- Docker: Provide working `Dockerfile` for both services
- Docs: Use **Swagger/OpenAPI** for both APIs

🐇 Optional Enhancements (only if time permits)

- Simulate **asynchronous processing** via **RabbitMQ** (e.g., log image audit info)
- Add **Elasticsearch & Logstash** for log indexing (basic integration only)

🗂️ Deliverables

Please provide the following in a **GitHub repo** (or zipped folder):

- ✅ Source code for both services
- ✅ `docker-compose.yml` to run the stack locally
- ✅ API documentation (Swagger or similar)
- ✅ Architecture diagram (Readme file with image or draw.io link)
- ✅ Sample requests/responses (via Postman or markdown)
- ✅ Description of your **security validations** and any design decisions
- ✅ (Optional) Postman collection or curl scripts