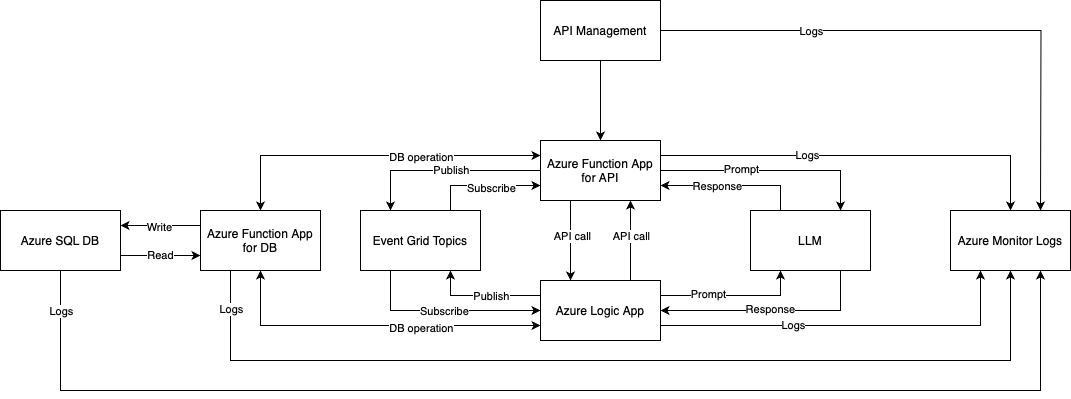
# Technical Skills - API Platform Architecture

### **Overview**

The following architecture represents a highly scalable, event-driven API platform implemented entirely with Azure serverless services. It integrates an LLM (Azure OpenAI) for enhanced capabilities, supports persistent storage, and includes observability via Azure Monitor Logs.



### **Component Descriptions**

* **API Management**: Acts as the secure, scalable entry point for external clients. Handles authentication, throttling, and routing to backend services.
* **Azure Function App for API**: Stateless compute layer that parses incoming API calls and initiates business logic orchestration. Scales automatically based on load.
* **Azure Logic App**: Orchestrates workflows. In this solution, one Logic App publishes a custom event to Event Grid, and another subscribes to that event and triggers LLM processing.
* **Event Grid Topics**: Enables decoupled communication between services. Events such as “task.process\_order” are published here to trigger downstream processing.
* **LLM (Azure OpenAI)**: Invoked by the subscriber Logic App to perform natural language understanding tasks such as summarization or classification. Accepts prompts dynamically based on input data.
* **Azure Function App for DB**: Handles structured write operations to Azure SQL DB. Ensures clean separation of concerns between business logic and data persistence.
* **Azure SQL DB**: Stores final processed results such as LLM outputs. Chosen for relational integrity and native Azure integration.
* **Azure Monitor Logs**: Aggregates logs and metrics from Functions, Logic Apps, and the database. Enables SLA/SLO monitoring and real-time diagnostics.

### **Workflow Description**

1. **Client API Request via API Management**The platform begins with a public-facing endpoint exposed through Azure API Management. It authenticates and routes incoming API requests to the backend.
2. **Request Handling by Azure Function App (API Layer)**API Management invokes a lightweight Azure Function App that parses and validates the request payload. It extracts user input and constructs an event schema for downstream processing.
3. **Workflow Orchestration via Logic App (Publisher)**The Function triggers Logic App, which publishes a structured event (e.g.: task.process\_order) to an Azure Event Grid Topic. This enables decoupled, event-driven architecture.
4. **Event Routing by Event Grid**Azure Event Grid routes the published event to a subscribing Logic App that handles LLM integration.
5. **LLM Task Orchestration via Logic App (Subscriber)**The subscribing Logic App receives the event, enriches the input, formats a dynamic prompt, and sends it to LLM (e.g., gpt-4o) via REST API.
6. **LLM Response Handling and Data Persistence**After receiving the LLM response (summary, classification, or analysis), the Logic App invokes another Azure Function App, which formats the result and writes it into an Azure SQL Database.
7. **Observability and Logging with Azure Monitor Logs**All Azure Functions, Logic Apps, and Event Grid events are instrumented to send logs into a centralized Azure Monitor Log service. This enables monitoring of API usage, execution success, and response times.

### **Scalability, Fault Tolerance, and High Availability**

* **Scalability**: All compute services (Functions, Logic Apps, Event Grid) scale automatically based on demand. SQL DB is provisioned with autoscaling tiers if required.
* **Fault Tolerance**: Logic Apps and Event Grid support retry policies. Functions can be configured with custom error handling and dead-letter queues.
* **High Availability**: Azure services are regionally and zonally redundant. No single point of failure exists in this fully decoupled, serverless design.

### **Serverless vs. Container-Base**

* **Serverless (used)**:
  + Azure Functions and Logic Apps reduce infrastructure management overhead.
  + Ideal for short-running, event-driven tasks like this LLM pipeline.
* **Container-based (not used here)**:
  + Suitable for long-running (such as large PDF scanning) or highly customized environments, but unnecessary for the current use case.

### **LLM Integration and Benefits**

* The subscriber Logic App generates a custom prompt and sends it to Azure OpenAI’s GPT model.
* This allows advanced NLP tasks (e.g., summarization, tagging, classification) to be automated.
* Benefits include:
  + Reduced manual workload
  + Intelligent data processing
  + Natural language interface for business processes

### **Infrastructure as Code**

All Azure resources are provisioned using Bicep templates. Each major resource (API Management, Function Apps, Logic Apps, Event Grid Topic, SQL DB, Azure Monitor) is defined in its own Bicep module:

param location string = resourceGroup().location

param sqlAdmin string = 'adminuser'

@secure()

param sqlPassword string

// API Management

resource apim 'Microsoft.ApiManagement/service@2021-12-01-preview' = {

name: 'gyg-apim'

location: location

sku: {

name: 'Consumption'

}

properties: {

publisherEmail: 'admin@gyg.com'

publisherName: 'GYG Company'

}

}

// App Service Plan (For Function Apps)

resource appPlan 'Microsoft.Web/serverfarms@2022-03-01' = {

name: 'gyg-app-plan'

location: location

sku: {

name: 'Y1'

tier: 'Dynamic'

}

kind: 'functionapp'

}

// Azure Function App for API

resource functionApi 'Microsoft.Web/sites@2022-03-01' = {

name: 'function-api'

location: location

kind: 'functionapp'

properties: {

serverFarmId: appPlan.id

}

}

// Azure Function App for DB

resource functionDb 'Microsoft.Web/sites@2022-03-01' = {

name: 'function-db'

location: location

kind: 'functionapp'

properties: {

serverFarmId: appPlan.id

}

}

// Azure SQL Server

resource sqlServer 'Microsoft.Sql/servers@2022-02-01-preview' = {

name: gyg-sql-server'

location: location

properties: {

administratorLogin: sqlAdmin

administratorLoginPassword: sqlPassword

}

}

// Azure SQL DB

resource sqlDb 'Microsoft.Sql/servers/databases@2022-02-01-preview' = {

name: gyg-sql-server/gyg-db'

location: location

dependsOn: [sqlServer]

}

// Event Grid Topic

resource eventTopic 'Microsoft.EventGrid/topics@2022-06-15' = {

name: gyg-topic'

location: location

}

// Logic App (combined publisher + subscriber as placeholder)

resource logicApp 'Microsoft.Logic/workflows@2019-05-01' = {

name: 'main-logic-app'

location: location

}

// Azure Monitor Logs (Log Analytics Workspace)

resource logs 'Microsoft.OperationalInsights/workspaces@2021-06-01' = {

name: 'log-workspace'

location: location

}

## **SLA and SLO Considerations**

### **SLA Definitions (Platform-Level Targets)**

| **Component** | **SLA** |
| --- | --- |
| Azure API Management | 99.95% |
| Azure Functions | 99.95% |
| Azure Logic Apps | 99.9% |
| Azure Event Grid | 99.99% |
| Azure SQL Database | 99.99% |
| Azure Monitor Logs | 99.9% |
| LLM (OpenAI) | 99.9% |

### **SLO Definitions (Custom, Measurable Objectives)**

| **Metric** | **Target** |
| --- | --- |
| API request availability | ≥ 99.9% over a 30-day period |
| API response time | ≤ 1 second |
| Function execution success rate | ≥ 99.95% |
| Event delivery latency | ≤ 200 ms (Event Grid to subscriber) |
| LLM response turnaround | ≤ 2 seconds (external dependency) |
| DB write latency | ≤ 500 ms (Azure Function to SQL DB) |

## **Set Up and Deploy the Solution**

### **Step-by-Step Deployment**

1. Log into Azure CLI via running “az login” in your Terminal or CMD
2. Create Resource Group: In command line, run: az group create --name gyg-platform-rg --location <region>
3. Deploy Bicep Template. E.g.: az deployment group create \ --resource-group gyg-platform-rg \ --template-file gyg-api=solution.bicep \ --parameters sqlPassword='gygpassword'
4. Set Up Function Code. Upload zipped function code to Azure portal (Azure Portal > Function App > Deployment Center)
5. Configure Logic App Workflows. Log into Azure portal and Create a “Logic App (Publisher)” and “Logic App (Subscriber)”

### **Test the API Platform**

1. Identify the input payload and expected result.
2. Send an API Request. Use Postman or any HTTP Client to send the payload to the endpoint according to the APi specification.
3. Observe Event Flow from the Log (Azure Monitor Logs).
4. Verify Output with expected output.
5. Monitor and Debug.