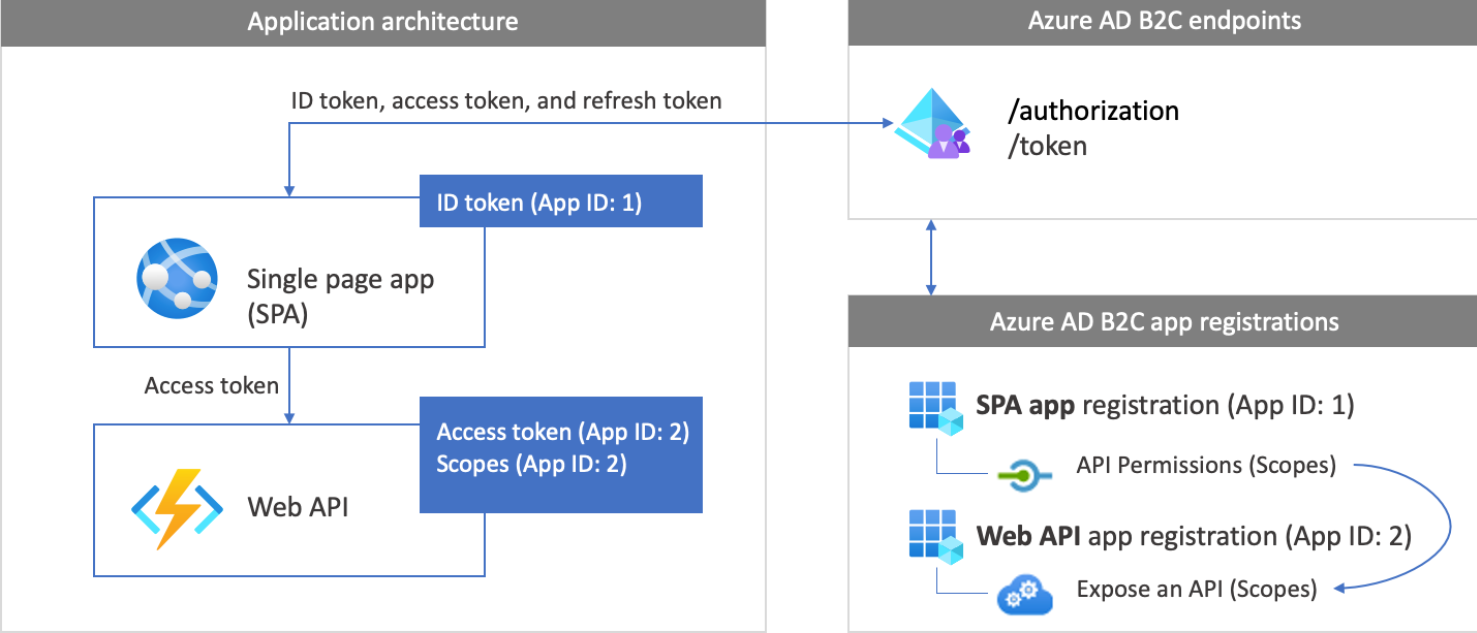
**Azure Microservices Architecture for Food Ordering Platform (Swiggy-like System)**

A diagram of a company

AI-generated content may be incorrect.



**💡 Use Case**

Swiggy-like food ordering platform with 100+ stores (APIs), real-time search, secure payments, and user personalization.

**🌟 Architecture Goals**

* Scalable and Event-Driven
* Fast, reliable, personalized Search
* Secure service-to-service communication
* Zero secrets in code
* Network-isolated microservices
* Full observability and traceability

**✅ End-to-End Architecture Flow**

**1️⃣ Authentication & API Gateway**

* Users authenticate via **Azure AD B2C / IdentityServer**.
* Requests pass through **Azure API Gateway** with JWT validation, routing, and throttling.

**Security:**

* OAuth 2.0 / OpenID Connect
* HTTPS enforced
* JWT bearer token validation
* Rate limiting and IP filtering

**2️⃣ Search Aggregator (Async Fan-Out)**

* **Azure Function** (timer/event-based) pulls from 100+ store APIs.
* Uses **User-Assigned Managed Identity** to retrieve secrets from **Azure Key Vault**.
* Data is stored in **Cosmos DB**, indexed by **Azure Cognitive Search**.
* Redis Cache is used for hot queries; Azure OpenAI enables semantic/personalized search.

**Security:**

* VNet-integration for Functions
* Cosmos DB, Redis, Azure Search, and Key Vault accessed via **Private Endpoints**.

**3️⃣ Cart, Booking, Discount APIs**

* Stateless APIs routed via API Gateway
* Deployed on App Service Plan or AKS
* Secure with JWT (external) and Managed Identity (internal)

**4️⃣ Order Processing**

* **Order API** captures order, generates Order ID & Correlation ID.
* Writes to **SQL Server** (Write DB)
* Publishes event to **Azure Service Bus**

**Patterns:**

* CQRS
* Circuit Breaker (Polly)
* Health checks + gRPC

**5️⃣ Payment Service (Multi-Gateway)**

* Consumes messages from Service Bus
* Interacts with Razorpay/Stripe/other providers
* Publishes PaymentSuccess or PaymentFailed to **Event Grid**

**Security:**

* Isolated subnet, outbound via NAT Gateway
* Secrets from Key Vault via Managed Identity

**6️⃣ Event-Driven Post-Processing**

* **Event Grid** triggers downstream services:
  + **Inventory Service** updates stock
  + **Notification Service** sends user alerts

**Benefits:**

* Fully decoupled
* Reliable
* Scalable

**🔐 Security Architecture**

|  |  |
| --- | --- |
| **Component** | **Security Features** |
| API Gateway | OAuth 2.0, IP Filtering, Rate Limiting |
| Azure Function | VNet-Integrated, Managed Identity |
| Cosmos DB | Private Endpoint, RBAC |
| Azure Search, Redis | Private Endpoint, NSG |
| Key Vault | Private Endpoint, Access via MSI |
| API1–API5 | Managed Identity, API Key, no public exposure |

**🌐 Network Architecture**

|  |  |
| --- | --- |
| Component | Design |
| VNet | Centralized Virtual Network |
| Subnets | App Subnet, Function Subnet, DB Subnet, Gateway Subnet |
| NSG | Enforces traffic rules at subnet level |
| ASG | Groups APIs/services for simplified NSG rule management |
| Private Endpoints | Used for Key Vault, Cosmos DB, Search, Redis |

**📊 Observability**

|  |  |
| --- | --- |
| Pillar | Tools |
| Logs | Serilog, App Insights |
| Metrics | Azure Monitor, Prometheus (opt) |
| Traces | App Insights Distributed Tracing |
| Alerts | Azure Monitor Alerts |
| Dashboards | Azure Workbooks, Grafana |

**🚀 Scalability**

|  |  |
| --- | --- |
| Component | Scale Capability |
| Azure Functions | Auto-scaling (consumption plan) |
| Cosmos DB | RU/s auto-scale, global distribution |
| Azure Search | Replica & shard scaling |
| Redis Cache | Clustered, high-throughput |
| App Services | Scale-out with rules or autoscaling |

“We built a secure, scalable, event-driven food-ordering architecture using Azure. All services are VNet-integrated and secured with Managed Identity and Key Vault. Cosmos DB is the aggregator store indexed by Azure Search, and Azure OpenAI personalizes results. The order-to-payment flow is CQRS-based and decoupled via Service Bus and Event Grid. Observability is achieved with Application Insights, Azure Monitor, and full correlation tracing. All sensitive services are behind Private Endpoints in isolated subnets.”

|  |  |
| --- | --- |
| 🔄 Async Aggregation | Azure Function + Cosmos DB = Clean fan-out/fan-in |

|  |  |
| --- | --- |
| 🚀 Search Layer | Azure Search + Redis Cache + OpenAI (Personalization) = Solid |

|  |  |
| --- | --- |
| 🔐 Secure Architecture | API Gateway + Azure AD B2C + Managed Identity + Key Vault = Excellent |

|  |  |
| --- | --- |
| 🧠 Scalable & Resilient | Durable components + Event Grid + Cosmos DB + Redis = High Availability |

|  |  |
| --- | --- |
| 🔄 Event-Driven Design | Order → Service Bus → Payment + Inventory + Notification = Good practice |

|  |  |
| --- | --- |
| 📦 CQRS / DDD-Friendly | Read/Write DB split (SQL + CosmosDB), gRPC, Health check, Correlation ID |

|  |  |
| --- | --- |
| ✅ Async Aggregator | Using Azure Function to pull from store APIs into Cosmos DB (Great!) |

|  |  |
| --- | --- |
| ✅ Cosmos DB | Excellent choice for flexible schema, fast reads, and global scaling |

|  |  |
| --- | --- |
| ✅ Azure Search | Right choice for enabling full-text, filtered, geo-based search |

|  |  |
| --- | --- |
| ✅ Secure Gateway | API Gateway with B2C/IdentityServer is standard and secure |

|  |  |
| --- | --- |
| ✅ Circuit Breaker | Included in Order API — resilience included |

|  |  |
| --- | --- |
| ✅ CQRS + Event Grid | Good separation of read/write and eventual consistency |

|  |  |
| --- | --- |
| ✅ Notification & Inventory | Decoupled with Service Bus and Event Grid (Nice!) |

|  |  |
| --- | --- |
| ✅ Azure OpenAI | Smart addition for search optimization or conversational UX |

Geo-Replication & Availability  
Cosmos DB supports multi-region write/read.  
Food ordering is a high-availability domain → Cosmos DB has 99.999% SLA.  
Ensures zero downtime even if one region fails.

Debugging **User-Assigned Managed Identity (UAMI)** when your **Search API or Azure Function** is calling downstream services (like API1) is a common scenario in secure microservice environments.

**✅ Scenario**

* **Search API (App Service)** or **Azure Function** uses **User-Assigned Managed Identity (UAMI)**.
* It calls **API1**, which is protected by Azure AD.
* You're debugging issues like:
  + Access denied (403)
  + Token not issued
  + Identity mismatch

**🔎 Step-by-Step: Debugging Managed Identity in Azure**

**✅ 1. Check Identity Assignment**

**🔧 App Service / Function App**

* Go to **Azure Portal → Your App → Identity → User Assigned**
* ✅ Ensure correct **UAMI is attached**

**🔧 API1 (the receiving end)**

* Go to **App Registrations** or **App/API App’s Access Policy**
* ✅ Ensure UAMI has proper **RBAC role or App Role** assigned

**✅ 2. Validate Token Issuance**

Use this to simulate token retrieval from the same identity:

bash

CopyEdit

# From inside Azure Cloud Shell or Function Console

curl "http://localhost:8080/msi/token/?resource=https://<your-api>.azurewebsites.net" \

-H "Metadata:true"

Or, in .NET:

csharp

CopyEdit

var credential = new ManagedIdentityCredential(clientId: "<your-uami-client-id>");

var token = await credential.GetTokenAsync(new TokenRequestContext(new[] { "https://<api1-domain>/.default" }));

✅ This lets you check:

* Is token being issued?
* Any errors like invalid\_client, unauthorized, AADSTS codes?

**✅ 3. Check Diagnostic Logs in Azure Monitor / App Insights**

* Enable **App Insights** on both **Search API** and **API1**
* Look for:
  + AuthorizationFailed
  + 403 - Access Denied
  + 401 - Unauthorized
  + Missing headers like Authorization: Bearer <token>

Use **KQL** in Log Analytics:

kusto

CopyEdit

requests

| where url contains "api1"

| where resultCode == "403" or resultCode == "401"

| project timestamp, url, resultCode, operation\_Name, customDimensions

**✅ 4. Use az CLI for Token Simulation**

To simulate a token as that identity:

bash

CopyEdit

az login --identity --username <UAMI-Client-ID>

az account get-access-token --resource https://<api1-app>.azurewebsites.net

Then use Postman or curl with:

http

CopyEdit

Authorization: Bearer <token>

✅ Useful for checking **manual call succeeds** with the token.

**✅ 5. Add Logging to Azure Function / API Client Code**

In your Function or .NET API:

csharp

CopyEdit

\_logger.LogInformation("Using client ID: {clientId}", managedIdentityClientId);

\_logger.LogInformation("Acquired token: {token.ExpiresOn}");

Also log any AccessTokenNotAvailableException or AAD exceptions explicitly.

**🔒 BONUS: Common Pitfalls to Check**

| **Issue** | **Resolution** |
| --- | --- |
| Wrong resource URI in token | Use correct audience (https://<api>.azurewebsites.net) |
| UAMI not assigned | Check identity blade under app/function |
| UAMI has no permissions | Assign proper **role (RBAC)** or app permission |
| Network issues (e.g., NSG, VNet) | Use Private Link Diagnostics |
| API1 not expecting token | Check **Authentication settings** on API1 |

**Deep-dive architecture + migration strategy questions** you’ll face in senior **Solution Architect / Enterprise Architect** interviews. Let’s break your response into **structured buckets** with STAR-style storytelling:

**✅ 1. Pre-Strategy Before Designing Architecture**

🟢 **Goal:** Build a secure, scalable, event-driven, multi-region ready architecture for a food-ordering platform involving 100+ partner store APIs.

**🔍 Pre-Assessment (Landing Zone + Discovery)**

* ✅ **Azure Landing Zone** set up for foundational governance:
  + Management Groups, Subscriptions (e.g., Dev, QA, Prod)
  + Azure Policy, Role-Based Access Control (RBAC)
  + Key Vault, Azure Monitor, Diagnostic Settings
  + VNet + NSG + Private Endpoints (Secure Network Topology)
* Assessed the following for each existing store API:
  + Tech stack (PHP, Node, .NET)
  + Data models (product, pricing, availability)
  + Auth mechanisms (API keys, OAuth2)
* Finalized a **hub-and-spoke model** VNet design

**✅ 2. Azure WAF Usage (Web Application Firewall)**

🛡 **WAF is critical to protect public-facing endpoints like API Gateway or App Gateway**

**🌐 Applied At:**

* Azure Application Gateway + Azure Front Door (if global)
* WAF policies applied for:
  + OWASP Top 10 rules
  + Rate limiting rules
  + IP whitelisting/blacklisting
  + Custom rules for bot protection

✅ *“WAF helps us filter malicious input, protect against DDoS, and enforce security compliance before traffic reaches our APIs.”*

**✅ 3. 6R Strategy (Migration Path)**

| **6R** | **How It Was Applied** |
| --- | --- |
| **Rehost** | APIs hosted in IIS or VM were moved to **App Service** with minimal change. |
| **Refactor** | APIs with tight coupling were refactored into **Azure Functions** or **App Services**. |
| **Replatform** | Some store APIs were containerized and moved to **AKS** with managed identity integration. |
| **Rearchitect** | Search logic was rearchitected from live synchronous to **asynchronous aggregator** using Azure Function + Cosmos DB. |
| **Retire** | Obsolete endpoints were deprecated after data sync. |
| **Retain** | Some on-prem store systems were retained and connected over hybrid VPN or Azure Arc. |

**✅ 4. Migration Tooling**

| **Task** | **Tool Used** |
| --- | --- |
| Azure Assessment | **Azure Migrate**, **App Insights** |
| Database Migration | **DMA** (Data Migration Assistant), **Azure DMS** |
| Infra as Code | **Terraform**, **Bicep**, **ARM Templates** |
| Secret Rotation | **Key Vault + Managed Identity** |
| API Testing & Load Test | **Postman + Azure Load Testing** |

**✅ 5. Why Azure Function (Aggregator)? Why not Logic Apps or AKS?**

| **Service** | **Why Used?** |
| --- | --- |
| **Azure Function** | Lightweight, serverless, cost-effective. Scales to 100+ store APIs. Timer/event-based model fits async aggregation. Supports MSI. |
| **Not Logic Apps** | Logic Apps = more visual but limited flow logic/custom retry control. |
| **Not AKS** | AKS adds operational overhead. Azure Function is enough for stateless fan-out. |

✅ *“I selected Azure Function for the search aggregator due to its simplicity, auto-scaling, VNet integration, and event-driven nature. It reduces operational cost and complexity.”*

**✅ 6. Multi-Region Architecture & Cosmos DB High Availability**

*“We were required to ensure the platform is resilient even if one Azure region fails.”*

**🌍 Cosmos DB Multi-Region Strategy**

* Cosmos DB used with **Multi-Region Write Enabled** (for write resiliency).
* Configured with:
  + **Region A** – Primary write
  + **Region B** – Secondary write + read fallback

**📈 Cosmos DB Benefits**

* SLA: **99.999% availability**
* Data replicated automatically in near real-time.
* **RTO (Recovery Time Objective)** = **<15 seconds**
* **RPO (Recovery Point Objective)** = **0 seconds**
  + **No data loss** due to automatic replication.

✅ *“In case Region A fails, Cosmos DB routes traffic automatically to Region B without downtime or data loss. Reads/writes continue with global distribution.”*

**✅ 7. Challenges & Resolutions**

| **Challenge** | **Resolution** |
| --- | --- |
| Inconsistent store APIs | Standardized data using Azure Function + data model mapping |
| Security concerns | Used API Gateway + Azure AD B2C + WAF + MSI |
| Token-based store API integration | Used Key Vault + Managed Identity |
| Regional failover | Cosmos DB multi-region write, Azure Front Door |
| Sensitive data | Encrypted at rest/in transit + role-based masking |
| Partner SLA mismatches (API1–API100) | Circuit Breaker + Retry policies with caching fallback |

**🧠 Final STAR Answer Summary (If Asked in Interview)**

**SITUATION**: We needed to build a secure, scalable, and resilient Azure-based architecture for a food-ordering platform involving 100+ external APIs and multi-region failover support.

**TASK**: Establish pre-migration strategy, cloud foundation, and secure, observable microservices architecture with real-time search and payments.

**ACTION**:

* Set up Azure Landing Zone with governance, NSGs, Private Endpoints, Key Vault, and WAF.
* Chose Azure Functions for async fan-out search aggregation using Cosmos DB + Azure Search + OpenAI.
* Implemented CQRS pattern, Event Grid, and service bus for resilient event flow.
* Used Managed Identity and Key Vault for secure integration, and Cosmos DB’s multi-region write with zero RPO.

**RESULT**:

* Achieved sub-second personalized search.
* Zero-downtime failover with Cosmos DB and Azure Front Door.
* Fully secure, observable, and automated CI/CD-ready environment.

**✅ 1. Services that Support & Should Use Private Endpoints**

These are **PaaS services** (Platform-as-a-Service) that don’t live inside your VNet but **can expose a Private Endpoint** inside your subnet.

| **Service** | **Private Endpoint** | **Why?** |
| --- | --- | --- |
| ✅ **Azure Key Vault** | ✅ Required | To access secrets from inside VNet securely, avoid public exposure |
| ✅ **Cosmos DB** | ✅ Required | Prevent public access to NoSQL store |
| ✅ **Azure Search** | ✅ Required | Lock search index access to internal services only |
| ✅ **Azure Redis Cache** | ✅ Recommended | Secure Redis with VNet + firewall rules |
| ✅ **Azure SQL DB** | ✅ Optional but Recommended | Secure your relational DB endpoint |

**✅ 2. Services That Live Inside VNet (Require Subnet Integration)**

These are **compute/services that are hosted in your VNet directly** or require VNet integration.

| **Service** | **Subnet Required?** | **Notes** |
| --- | --- | --- |
| ✅ **App Services (APIs)** | ✅ via VNet Integration | To reach private endpoints (Key Vault, Cosmos, etc.) |
| ✅ **Azure Functions** | ✅ via Premium Plan or App Plan | Needed to call private services (Cosmos, Redis) |
| ✅ **Application Gateway** | ✅ Lives in subnet | Used with WAF; frontend to APIs |
| ✅ **Azure Front Door** | ❌ Global CDN | Does not live in your VNet |
| ✅ **AKS Nodes** | ✅ Lives in subnet | If used in future for scaling APIs |
| ✅ **Self-hosted VM / Worker** | ✅ Yes (if used) | Requires subnet + NSG |

**🎯 Summary Table: Your Architecture Specific**

| **Component** | **Private Endpoint** | **VNet Subnet Needed** | **Why?** |
| --- | --- | --- | --- |
| API Gateway (APIM / Ocelot) | ❌ Public (secured via WAF) | ✅ Yes (App Gateway subnet) | Entry point, protected via WAF |
| Azure Functions | ❌ (PE not needed) | ✅ Yes | Needs subnet to access internal services via PE |
| Cosmos DB | ✅ Yes | ❌ (external PE) | Private IP exposed into your subnet |
| Azure Redis | ✅ Yes | ❌ (external PE) | Access via private IP only |
| Azure Key Vault | ✅ Yes | ❌ (external PE) | Managed Identity + private DNS zone |
| Azure Search | ✅ Yes | ❌ (external PE) | Secured internal search access only |
| SQL Server (Write DB) | ✅ Optional | ❌ (external PE) | Use if not hosted on VM |
| Event Grid / Service Bus | ❌ No | ❌ No | Secure with RBAC, doesn't support PE |
| Notification/Inventory Services | ❌ No (if internal) | ✅ Yes (App Subnet) | Host in same subnet or App Plan |

**🔐 Best Practices Checklist**

* ✅ Use **Private Endpoints** for every PaaS service storing or processing data.
* ✅ Use **VNet Integration** for all compute (Function, App Services) that access private endpoints.
* ✅ Use **NSGs + ASGs** to restrict East-West traffic.
* ✅ Use **Private DNS Zone** to resolve \*.privatelink.\* endpoints.