Top System Design, Azure, DevOps, and Microservices Questions for .NET Core Developers

---

1. How do you implement Microservices architecture in .NET Core?

Answer:

To implement Microservices in .NET Core:

✅ **Independent Services** – Each microservice is a separate ASP.NET Core Web API project.
✅ **Communication** – Use **REST APIs** (HTTP) or **gRPC** (faster, binary protocol) for sync calls. For async messaging, use **Azure Service Bus** or **RabbitMQ**.
✅ **Database per Service** – Each microservice has its own database (SQL, Cosmos DB, etc.) to avoid tight coupling.
✅ **API Gateway** – Use **Ocelot** (for lightweight gateways) or **Azure API Management** (enterprise-grade) to route requests.
✅ **Service Discovery** – Tools like **Consul** or **Kubernetes DNS** help services find each other dynamically.
✅ **Deployment** – Containerize using **Docker** and deploy on **Kubernetes (AKS)** for scalability.

🔷 **Why Microservices?** – Scalability, independent deployments, fault isolation.

---

2. How is authentication handled using JWT in .NET Core?

Answer:

JWT (JSON Web Token) is used for stateless authentication:

✅ **Flow:**

1. User logs in → Server validates credentials → Returns a **JWT token**.
2. Client sends this token in the **Authorization header** for subsequent requests.

✅ **Token Structure:**

- **Header** (Algorithm, token type)
- **Payload** (Claims like user ID, roles, expiry)
- **Signature** (Verifies token integrity)

✅ **Implementation in .NET Core:**

csharp

```
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options => {
```

```
    options.TokenValidationParameters = new TokenValidationParameters {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidIssuer = "yourIssuer",
        ValidAudience = "yourAudience",
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("yourSecretKey"))
    };
});
```

✅ **Best Practices:**

- Use **short-lived tokens** + refresh tokens.
- Store tokens securely (**HTTP-only cookies** or **localStorage with HTTPS**).
- Always validate **issuer, audience, and expiry**.

---

3. What is an Azure Function and when would you use it?

Answer:

Azure Functions are **serverless** compute services that run code in response to events.

✅ **When to Use?**

- **Event-driven tasks** (e.g., process a file when uploaded to Blob Storage).
- **HTTP APIs** (serverless backend).
- **Scheduled jobs** (e.g., daily report generation).
- **Lightweight microservices** (cost-effective, no server management).

✅ **Triggers:**

- HTTP requests
- Blob Storage changes
- Queue messages (Service Bus, Storage Queue)
- Timers (CRON jobs)

🔹 **Example:** Send an email when a new user registers (triggered by a queue message).

---

4. What is Azure Service Bus?

Answer:

Azure Service Bus is a **cloud messaging service** for decoupling applications.

✅ **Features:**

1. **Queues** – Point-to-point messaging (one sender, one receiver).
2. **Topics & Subscriptions** – Publish-subscribe model (one sender, multiple subscribers).

✅ **Use Cases:**

- Microservices communication (async, reliable).
- Order processing (queues ensure no orders are lost).
- Event-driven architectures (e.g., notify multiple services when an order is placed).

✅ **Why Not REST API?** – Service Bus ensures **message durability, retries, and scalability**.

---

5. How does CI/CD work in Azure DevOps?

Answer:

CI/CD automates **building, testing, and deploying** code.

✅ **CI (Continuous Integration)** – On every Git commit:

- Code is built.
- Unit tests run.
- Artifacts (e.g., Docker images) are published.

✅ **CD (Continuous Deployment)** – Automatically deploys to **Dev → Stage → Prod**.

✅ **Sample Azure Pipeline (YAML):**

yaml

```
trigger:
  branches: [ main ]

pool: vmImage: 'windows-latest'

steps:
- task: DotNetCoreCLI@2
  inputs:
    command: 'build'
    projects: '**/*.csproj'
- task: DotNetCoreCLI@2
```

```
inputs:
  command: 'test'
```

✅ **Key Steps:**

1. **Restore packages** (dotnet restore)
2. **Build** (dotnet build)
3. **Test** (dotnet test)
4. **Publish** (dotnet publish)
5. **Deploy** (to Azure App Service, AKS, etc.)

---

6. How do you handle distributed load and monitoring in microservices?

Answer:

✅ **Handling Load:**

- **Load Balancer** (Azure Load Balancer, Application Gateway).
- **Auto-scaling** (Kubernetes/AKS scales pods based on CPU/memory).
- **Multi-region deployment** (Azure Traffic Manager for failover).

✅ **Monitoring:**

- **Azure Application Insights** – Logs, performance tracking, alerts.
- **Azure Monitor** – Metrics (CPU, memory, response times).
- **Prometheus + Grafana** (for Kubernetes monitoring).
- **Correlation IDs** – Track requests across microservices.

🔹 **Why Monitoring?** – Detect failures, optimize performance, debug issues.

🚀 **Master these, and you'll ace your .NET Core interviews!**

#DotNetCore #Azure #Microservices #DevOps #TechInterviews #SystemDesign