.NET CORE MICROSERVICES

100+ .Net Microservices

Interview Questions and Answers

Sai Reddy 5/16/25 Interview Tips

100+ .Net Microservices Interview Q&A

.Net Microservices interview question (0–2 years level)

1. What are microservices?

Microservices are a software architecture style where applications are built as small, independent services.
Each service handles a specific business function and communicates via APIs.
They are independently deployable and scalable.
Improves maintainability and flexibility,.
Example: **An e-commerce app has separate services for Orders, Products, Payments.**

2. Difference between monolithic and microservice architecture?

Monolithic apps are single codebases, harder to scale or modify.
Microservices split functions into independent modules.
Monoliths share one database; microservices often use separate ones.
Microservices allow team independence and better fault isolation.
Example: Monolith = single .dll; Microservices = multiple APIs for cart, orders, etc.

3. Role of API Gateway in microservices?

API Gateway acts as a single entry point for all microservices.
It handles routing, security, rate limiting, etc.
Reduces direct client-to-service communication.
Helps with load balancing and response aggregation.
Example: Ocelot in .NET routes requests to the right backend services.

4. What is a service registry (e.g., Consul or Eureka)?

Service registry tracks all available microservices and their locations.
Used for service discovery in dynamic environments.
Helps microservices find each other without hardcoded URLs.
Automatically updates registry when services start or stop.
Example: Eureka client registers a service so others can call it.

5. What is REST and how is it used in microservices?

REST is an architectural style using HTTP to communicate.
Each microservice exposes endpoints (URLs) to perform CRUD operations.
Uses verbs like GET, POST, PUT, DELETE.
Lightweight and stateless communication.
Example: GET /api/products returns a product list.

6. How do microservices communicate with each other?

Microservices talk via HTTP (REST), gRPC, or messaging queues.
Communication can be synchronous or asynchronous.
Each method depends on the business need and performance.
Asynchronous is preferred when high decoupling is needed.
Example: Product service calls Order service via HttpClient.

Sai Reddy
saireddy-dotnetfs

7. What is the use of Swagger in microservices?
Swagger generates interactive API documentation.
Helps test endpoints and understand request/response formats.
Great for developers and external consumers of your APIs.
Supports code generation too.
Example: Swagger UI shows /api/orders with parameters and test button.

8. What is Docker and why is it used in microservices?
Docker packages apps into containers with all dependencies.
Helps run services consistently across environments.
Improves deployment, isolation, and scalability.
Supports CI/CD and faster testing.
Example: A Dockerfile creates a container for the Product service.

9. What is the purpose of a load balancer?
Distributes incoming traffic across multiple service instances.
Ensures no single instance is overwhelmed.
Improves availability and scalability.
Can be software-based (NGINX) or hardware-based.
Example: Azure Load Balancer routes requests to healthy service instances.

10. What is HTTP vs HTTPS? Why is HTTPS important?
HTTP is unsecured; data is sent in plain text.
HTTPS encrypts data using SSL/TLS.
Protects against man-in-the-middle attacks.
HTTPS is essential for secure microservice communication.
Example: Use https://api.example.com instead of http://.

11. What is dependency injection in .NET Core?
It's a design pattern to inject required objects at runtime.
Improves testability, maintainability, and loose coupling.
.NET Core has built-in support via Startup.cs.
Used for services, repositories, config, etc.
Example: Inject IOrderService into controller constructor.

12. How do you call another microservice in .NET?
Use HttpClient or HttpClientFactory for REST calls.
Can also use message queues for async communication.
Use proper error handling and retries.
Prefer using typed clients for better structure.
Example: httpClient.GetAsync("http://orderservice/api/order/1")

13. What is an HttpClientFactory in .NET Core?
Manages and reuses HttpClient instances properly.
Avoids socket exhaustion issue from frequent instantiation.
Supports named/typed clients and policies.

Improves testability and configuration.
Example: services.AddHttpClient<IProductClient, ProductClient>()

14. What is appsettings.json used for?
Stores application configuration like DB strings, keys, URLs.
Supports environment-specific files like appsettings.Development.json.
Automatically loaded at startup.
Can bind to POCO classes using options pattern.
Example: "ConnectionStrings": { "Default": "..." }

15. What is the role of a message queue (e.g., RabbitMQ)?
Used for async communication between services.
Decouples sender and receiver.
Improves fault tolerance and scalability.
Stores messages until processed.
Example: Order service sends a message to Inventory queue.

16. How do you handle configuration in microservices?
Use appsettings.json, environment variables, or config services.
Each service should have its own configuration.
Centralized config like Azure App Configuration can be used.
Secrets should be stored securely (e.g., Azure Key Vault).
Example: Inject options from config file using IOptions<T>.

17. How can you version an API in ASP.NET Core?
Use URL path (/api/v1/orders), query string, or headers.
Helps avoid breaking changes for old clients.
Use Microsoft.AspNetCore.Mvc.Versioning package.
Controllers can be decorated with [ApiVersion("1.0")].
Example: [Route("api/v1/[controller]")]

18. What is Health Check in microservices?
Monitors the availability of a service.
Exposes a health endpoint (e.g., /health) for tools to ping.
Returns service status (healthy, degraded, etc.).
Integrated with Kubernetes, load balancers, etc.
Example: services.AddHealthChecks() and use middleware.

19. What are the benefits of using microservices?
Independent deployment and scaling.
Better fault isolation and maintainability.
Technology flexibility for each service.
Easier for multiple teams to work in parallel.
Example: Only updating the Billing service doesn't affect others.

20. Difference between synchronous and asynchronous communication?

Synchronous: Caller waits for response (e.g., HTTP call).
Asynchronous: Caller continues without waiting (e.g., queue).
Async increases performance and decoupling.
Sync is simpler but tightly coupled.
Example: Email sent via queue (async), vs direct API call (sync).

21. **Write main components of Microservices.**

Some of the main components of microservices include:

- Containers, Clustering, and Orchestration
- IaC [Infrastructure as Code Conception]
- Cloud Infrastructure
- API Gateway
- Enterprise Service Bus
- Service Delivery

22. **Explain PACT in microservices.**

PACT is defined as an open-source tool that allows service providers and consumers to test interactions in isolation against contracts that have been made to increase the reliability of microservice integration. It also offers support for numerous languages, such as Ruby, Java, Scala, .NET, JavaScript, Swift/Objective-C.

23. **What is the main role of docker in microservices?**

Docker generally provides a container environment, in which any application can be hosted. This is accomplished by tightly packaging both the application and the dependencies required to support it. These packaged products are referred to as Containers, and since Docker is used to doing that, they are called Docker containers. Docker, in essence, allows you to containerize your microservices and manage these microservices more easily.

.NET microservices (3–7 years)

1. How do you handle service discovery in microservices?

Service discovery enables microservices to locate each other dynamically.
You can use tools like **Eureka**, **Consul**, or Kubernetes **DNS-based discovery**.
In .NET, discovery can be configured via **Ocelot or Steeltoe** libraries.
Removes hardcoded URLs and supports scaling.
Example: Product service registers in Consul and Order service queries Consul for its address.

2. What is the circuit breaker pattern? How do you implement it in .NET?

Circuit breaker prevents cascading failures when a service is down.
It stops calls temporarily and retries after a cooldown.

In .NET, use **Polly** to implement this pattern.
Enhances resilience and system stability.
Example: If Payment API fails 3 times, Polly opens the circuit for 30 seconds.

3. Role of Ocelot in microservices architecture?
Ocelot is a .NET API Gateway library.
It handles request routing, load balancing, rate limiting, and authentication.
Acts as a single entry point to your microservices.
Reduces client complexity and enforces policies centrally.
Example: GET /orders on Ocelot routes to http://orderservice/api/orders.

4. How do you handle security (authentication/authorization) in microservices?
Use **OAuth2/JWT tokens** for authentication.
Centralize security with an **Identity Provider (e.g., IdentityServer4 or Azure AD)**.
Services validate tokens locally for authorization.
Avoid storing credentials in services.
Example: User logs in via Auth service → gets JWT → includes token in requests.

5. Common challenges in microservices?
- Service communication (sync/async).
- Data consistency and duplication.
- Deployment complexity.
- Debugging and tracing across services.
- Network latency and security.

Example: Hard to track a failed request spanning 3 microservices without tracing.

6. How do you manage data consistency across microservices?
Use **event-driven architecture** and **eventual consistency**.
Avoid distributed transactions.
Use messaging tools like RabbitMQ or Kafka.
Design each service to manage its own data.
Example: After placing an order, an event triggers inventory deduction.

7. What is eventual consistency?
It means all services will reach a consistent state **over time**.
Common in distributed systems.
Ensures availability and partition tolerance (CAP Theorem).
Used in event-driven or message-based systems.
Example: After a payment, the order status updates a few seconds later.

8. How do you handle logging in microservices?
Use **structured logging** (e.g., Serilog) in each service.
Include correlation IDs to trace requests across services.

Send logs to a **centralized log store**.
Log both errors and key business events.
Example: Log request IDs in both Order and Payment services.

9. What is centralized logging? Which tools have you used?
All microservices send logs to a **single platform** for analysis.
Tools: **ELK stack (Elasticsearch, Logstash, Kibana)** or **Seq**, **Grafana Loki**.
Helps in debugging and monitoring.
Supports searching logs across services.
Example: All microservices push logs to Elasticsearch; view in Kibana dashboard.

10. How does distributed tracing work?
Tracks a request across multiple microservices.
Adds **trace IDs** and **span IDs** to requests.
Tools: **Jaeger**, **Zipkin**, **OpenTelemetry**.
Helps diagnose latency and errors across services.
Example: See timeline of a single order flowing through 4 services.

11. How to deploy microservices using Docker?
Create **Dockerfiles** for each microservice.
Build images and run them using docker run or docker-compose.
Isolate services with their dependencies.
Can scale independently and port anywhere.
Example: docker-compose up starts Product, Order, and Auth services together.

12. What is Kubernetes and how does it help microservices?
Kubernetes automates deployment, scaling, and management of containers.
Manages service discovery, load balancing, and health checks.
Supports rolling updates and self-healing.
Ideal for large-scale microservices systems.
Example: Kubernetes automatically restarts a crashed Order service pod.

13. How do you monitor microservices?
Use tools like **Prometheus**, **Grafana**, **App Insights**, or **Datadog**.
Track CPU, memory, latency, error rates, etc.
Enable health checks and alerts.
Expose custom metrics using libraries like App.Metrics.
Example: Monitor GET /products response times in Grafana dashboard.

14. What is the Saga pattern? Where is it used?
Saga handles **distributed transactions** across services.
Each step is a local transaction, followed by a compensating action if it fails.
Can be orchestration-based or choreography-based.

Used in order-processing, booking systems, etc.

Example: If payment fails, Saga triggers cancel shipment and refund inventory.

15. How do you test microservices?

- Unit Testing: Logic inside services
- Integration Testing: External dependencies
- Contract Testing: APIs between services
- End-to-End Testing: Full workflow

  Use mocks/stubs for isolation.

  Example: Test that placing an order triggers inventory and email services.

16. How do you manage secrets and environment-specific configurations?

Use environment variables or secure vaults.

Tools: **Azure Key Vault**, **AWS Secrets Manager**, **HashiCorp Vault**.

Avoid storing secrets in code or config files.

Use dependency injection to load config.

Example: Read DB password from Azure Key Vault during app startup.

17. What are bounded contexts?

It defines a clear **boundary** around a business domain.

Each microservice handles a single bounded context.

Helps maintain **clean domain models** and reduces coupling.

Matches with **DDD (Domain-Driven Design)**.

Example: Order and Inventory are separate bounded contexts with their own data models.

18. How do you migrate a monolith to microservices?

Start with identifying bounded contexts.

Extract one service at a time (e.g., auth or order).

Use APIs/events to integrate with the monolith.

Ensure backward compatibility.

Example: Move product catalog logic into a new microservice with its own DB.

19. How do you ensure resiliency and fault tolerance?

Use **circuit breakers**, **retries**, **timeouts**, **fallbacks** (via Polly).

Design for **graceful degradation**.

Use load balancers, health checks, and service replicas.

Implement async messaging to decouple.

Example: If shipping service fails, show "pending" status to the user.

20. How do you manage versioning in microservices?

Use **URL-based** (/v1/orders), **header-based**, or **query string**.

Maintain backward compatibility for old clients.

Keep separate controllers or service endpoints per version.

Deprecate gradually.

Example: Support /api/v1/products and /api/v2/products with different response shapes.

.NET microservices (7–14 years experience level):

1. How would you design a large-scale microservices architecture from scratch?

Start with defining business domains and bounded contexts.

Design independent, loosely coupled services with separate data stores.

Use API Gateway, service discovery, and centralized logging/tracing.

Plan for scalability, resilience, and deployment automation.

Example: An e-commerce system with services for orders, payments, inventory, each independently deployable.

2. What are domain-driven design principles in microservices?

Focus on business domains to define service boundaries.

Use ubiquitous language shared by developers and domain experts.

Apply bounded contexts to isolate models and data.

Ensure services own their data and logic independently.

Example: Separate Order domain from Customer domain with own databases.

3. How do you decide service boundaries in microservices?

Analyze business capabilities and domain models.

Identify areas with high cohesion and low coupling.

Consider data ownership and transaction boundaries.

Use DDD bounded contexts and team structure as guides.

Example: Payments and shipping are separate services due to distinct workflows.

4. How do you implement CQRS in .NET?

Separate read and write models using distinct classes and data stores.

Use commands to change state and queries to read data.

Leverage MediatR or custom messaging for commands and queries.

Improve scalability and simplify complex domain logic.

Example: Commands update the database, queries read from a read-optimized cache.

5. Pros and cons of microservices vs modular monolith?

Microservices offer independent deployability and scalability but add complexity.

Modular monoliths are easier to develop and debug but less flexible for scaling.

Microservices require more infrastructure and automation.

Modular monoliths are good for smaller teams or projects.

Example: Early startup may start with modular monolith, then evolve to microservices.

6. How do you implement observability (metrics, logs, tracing)?

Collect structured logs with correlation IDs.

Expose metrics via Prometheus and visualize with Grafana.

Use distributed tracing tools like Jaeger or OpenTelemetry.

Enable health checks and alerting.

Example: Trace request flow across Order, Payment, and Shipping services.

7. Approach to inter-service communication – sync vs async?

Use synchronous HTTP/gRPC for low-latency, simple interactions.

Use asynchronous messaging for decoupling and resilience.

Avoid blocking calls in critical paths.

Choose based on business requirements and system load.

Example: Order service calls Payment service synchronously but updates Inventory asynchronously.

8. How do you handle failover and retries in microservices?

Implement retries with exponential backoff using Polly.

Use circuit breakers to prevent cascading failures.

Employ fallback strategies for graceful degradation.

Deploy redundant instances with load balancing.

Example: Retry Payment API call 3 times before opening circuit breaker.

9. How do you implement blue-green or canary deployments?

Blue-green runs two identical production environments; switch traffic instantly.

Canary releases deploy new versions to a subset of users first.

Use Kubernetes or Azure DevOps pipelines to automate.

Monitor metrics to roll back if needed.

Example: Deploy new API version to 10% users, monitor errors, then full rollout.

10. Experience with service meshes like Istio?

Istio provides traffic management, security, and observability at the service level.

Supports mTLS encryption, retries, circuit breakers, and policy enforcement.

Simplifies microservices networking without changing code.

Useful for large, complex microservices ecosystems.

Example: Use Istio to enforce secure communication between Payment and Order services.

11. Dealing with eventual consistency and distributed transactions?

Avoid distributed ACID transactions; rely on eventual consistency.
Use Saga pattern or event-driven workflows.
Ensure idempotent event handlers and compensating transactions.
Accept some delay in data synchronization for availability.
Example: Update order status after payment event confirms success.

12. Explain your CI/CD pipeline for microservices.

Automate build, test, and deployment using pipelines (Azure DevOps, Jenkins).
Use containerization (Docker) and orchestration (Kubernetes).
Include unit, integration, and contract testing.
Deploy microservices independently with versioning.
Example: Commit triggers build → Docker image push → Kubernetes deployment rollout.

13. Anti-patterns in microservices design?

Creating tightly coupled services sharing the same database.
Over-splitting services causing operational overhead.
Ignoring monitoring and logging.
Using synchronous communication for all interactions.
Example: A microservice directly querying another service's database breaks independence.

14. Ensuring high availability and scalability?

Deploy multiple instances behind load balancers.
Use auto-scaling in Kubernetes or cloud platforms.
Implement health checks and failover mechanisms.
Use stateless services for easy scaling.
Example: Kubernetes pod replicas auto-scale based on CPU usage

15. Enforcing API security across multiple services?

Centralize authentication using OAuth2/OpenID Connect.
Use JWT tokens for stateless authorization.
Apply API Gateway policies for rate limiting and IP whitelisting.
Encrypt sensitive data in transit and at rest.
Example: Ocelot Gateway validates JWT before forwarding requests.

16. Managing backward compatibility in APIs?

Use versioning in URLs or headers.
Deprecate old endpoints gradually.
Avoid breaking changes; add new fields as optional.
Test old clients against new versions.
Example: Maintain /api/v1/orders while introducing /api/v2/orders.

17. Governance practices in microservices?

Enforce coding standards and documentation.

Use centralized logging and monitoring.

Automate security scanning and compliance checks.

Establish service ownership and lifecycle policies.

Example: Mandatory code reviews and automated security scans in CI pipeline.

18. Handling data replication and synchronization?

Use event-driven updates and publish-subscribe patterns.

Leverage CDC (Change Data Capture) tools when applicable.

Accept eventual consistency for distributed systems.

Avoid synchronous cross-service data writes.

Example: Inventory service updates triggered by order events via Kafka.

19. Experience with event sourcing in .NET?

Store state changes as a sequence of events instead of current state.

Rebuild state by replaying events.

Useful for audit trails and complex workflows.

Libraries: **EventStoreDB**, **Marten**.

Example: Order service saves each order change event, reconstructs order history.

20. Measuring performance and optimizing bottlenecks?

Profile services with tools like **dotTrace** or **Application Insights**.

Analyze logs and traces to find slow operations.

Optimize database queries, caching, and parallel processing.

Load test to identify capacity limits.

Example: Detect slow payment validation calls and cache frequent results.


Best Wishes
for the Interview!
Be confident and believe
in yourself.