Komunikacja w systemach rozproszonych

REST · gRPC · Message Brokers · API Gateway

mgr inż. Jakub Woźniak

Politechnika Poznańska · Instytut Informatyki · Semestr letni 2025/26

Plan wykładu

Agenda

- Komunikacja synchroniczna: REST, gRPC, GraphQL
- Komunikacja asynchroniczna: Message Brokers
- Wzorce komunikacji
- API Gateway i Backend for Frontend
- Ewolucja kontraktów i idempotencja
- ☁ Managed messaging i managed API Gateway

Komunikacja synchroniczna

Cechy REST

- Zasoby identyfikowane przez URL (`/api/orders/123`)
- Operacje przez metody HTTP (GET, POST, PUT, DELETE)
- Bezstanowość — każdy request zawiera pełny kontekst
- Format: najczęściej JSON

REST — zalety i wady

Zalety:

- Uniwersalność — każdy język
- Cacheowalność (HTTP, CDN)
- Prostota debugowania

Wady:

- Over/under-fetching
- Brak formalnego kontraktu (opcjonalny OpenAPI)
- Tekstowy JSON — większy payload

Cechy gRPC

- **Protocol Buffers** — binarny format ze schematem
- Transport: **HTTP/2** — multiplexing, kompresja nagłówków
- Generowanie kodu z pliku `.proto`
- Natywny streaming (unary, server, client, bidirectional)

gRPC — zalety i wady

Zalety:

- 10× mniejszy payload
- 2–5× mniejsze opóźnienie
- Formalny kontrakt
- Natywny streaming

Wady:

- Nie działa w przeglądarce (bez grpc-web)
- Trudniejsze debugowanie (dane binarne)
- Wymaga generowania kodu

Cechy GraphQL

- Klient **definiuje kształt odpowiedzi**
- Jeden endpoint (`/graphql`)
- Silne typowanie, introspection

Kiedy stosować:

- Złożone, zagnieżdżone dane
- Wielu klientów (mobile vs web)
- Szybko iterujący frontend

Kiedy nie:

- Proste CRUD — REST wystarczy
- Operacje zapisu
- Ryzyko ciężkich zapytań

Porównanie: REST vs gRPC vs GraphQL

Cecha	REST	gRPC	GraphQL
Format	JSON	Protobuf	JSON
Transport	HTTP/1.1+	HTTP/2	HTTP/1.1+
Kontrakt	OpenAPI	`.proto`	Schema
Streaming	SSE	Natywny	Subskrypcje
Typowe użycie	Publiczne	Serwis↔serwis	Frontend↔back.
Opóźnienie	Średnie	Niskie	Średnie

Komunikacja asynchroniczna

Message Brokers — idea

Idea

Producent wysyła wiadomość do **brokera**, konsument ją odbiera. Nie muszą się znać ani działać jednocześnie.

Queue (punkt-punkt)

Wiadomość → **jeden** konsument.

- Podział pracy
- SQS, RabbitMQ

Topic (pub/sub)

Wiadomość → **wszyscy** subskrybenci.

- Powiadamianie zdarzeniami
- Kafka, SNS, NATS

Kluczowe koncepcje

- **Topic** — nazwany strumień wiadomości
- **Partition** — shard topiku, ordering w obrębie partycji
- **Consumer Group** — każda partycja → jeden konsument w grupie
- **Retention** — wiadomości nie znikają po konsumpcji

Kafka ≠ tradycyjna kolejka:

- Konsumenci mogą „cofnąć się w czasie“ (replay)
- Append-only log — architektura oparta na logu

RabbitMQ vs Kafka vs NATS

Cecha	RabbitMQ	Kafka	NATS
Model	Queue + Pub/Sub	Log (Pub/Sub)	Pub/Sub + Queue
Ordering	Per queue	Per partition	Brak gwarancji
Replay	Nie	Tak	JetStream: Tak
Przepustowość	Dziesiątki tys/s	Setki tys/s	Miliony/s
Złożoność ops	Średnia	Wysoka	Niska

Wzorce komunikacji

Request-Response i Fire-and-Forget

Request-Response

Klient wysyła, czeka na odpowiedź.

- REST, gRPC (unary)
- **1.** Prosty model
- – Tight coupling

Fire-and-Forget

Producent wysyła, nie czeka.

- Kolejka, zdarzenie
- **1.** Loose coupling
- – Brak potwierdzenia

Request-Reply via Queue i Event Notification

Request-Reply via Queue

Żądanie i odpowiedź przez kolejki.

- Correlation ID łączy parę
- **1.** Async + odpowiedź
- – Złożoność

Event Notification

„Coś się stało“ — konsumenci decydują.

- `OrderPlaced`, `PaymentDone`
- **1.** Maks. decoupling
- – Trudne śledzenie

API Gateway

Funkcje API Gateway

API Gateway — bramka do systemu

- **Trasowanie** — kierowanie żądań do serwisów
- **Rate limiting** — ochrona przed przeciążeniem
- **Uwierzytelnianie** — weryfikacja tokenów na brzegu
- **Load balancing** — rozproszenie ruchu

Narzędzia self-hosted: Kong, Envoy, Traefik

Backend for Frontend (BFF)

Wzorzec BFF

Dedykowany gateway **per typ klienta**: mobile (mniejsze payloady), web (bogatsze odpowiedzi), IoT.

Dlaczego nie jeden gateway?

- Różni klienci = **różne potrzeby**
- Zespół mobilny nie blokuje zespołu webowego
- Unikamy „God Gateway“ z nadmierną logiką

Ewolucja kontraktów

API Versioning

Strategie

- **URL path**: `/api/v1/orders`
- **Header**: `Accept: ...v2+json`
- **Query**: `?version=2`

Zasada nr 1

Nigdy nie łam backward compatibility.

Dodawanie pól = OK.

Usuwanie/zmiana = BREAKING.

Schema Evolution

Serializacja ze schematem

- **Avro** — schema w rejestrze, ewoluowalny
- **Protocol Buffers** — pola numerowane, backward/forward compatible
- **JSON Schema** — walidacja JSON

Zasady:

- Nowe pola = **opcjonalne** z wartością domyślną
- Nigdy nie zmieniaj numeru/nazwy istniejącego pola
- Usuwanie = `reserved`

Idempotencja

Idempotencja operacji

Definicja

Operacja jest **idempotentna**, jeśli wykonanie jej raz daje ten sam efekt co wielokrotne wykonanie.

Z natury idempotentne

- `GET /orders/123`
- `PUT /orders/123 {...}`
- `DELETE /orders/123`

NIE idempotentne

- `POST /orders` — nowe zamówienie
- `POST /payments` — pobiera pieniądze!

Klucze idempotentności

Rozwiązanie dla POST

Klient generuje unikalny **Idempotency-Key** w nagłówku. Serwer: sprawdza w Redis → przetworzone? zwróć zapisaną odpowiedź : przetwórz i zapisz.

Dlaczego to krytyczne:

- Timeout ≠ „request nie dotarł“
- Retry bez idempotencji = **duplikaty**
- Stripe, PayPal, AWS — wszystkie mają `Idempotency-Key`

Sync vs Async

Macierz decyzyjna

Scenariusz	Sync	Async
Użytkownik czeka na odpowiedź	Tak	Nie
Operacja trwa > 1s	Nie	Tak
Wielu odbiorców zdarzenia	Nie	Tak
Proste CRUD	Tak	Nie
Event-driven workflow	Nie	Tak

Cloud messaging i managed API Gateway

Managed Message Brokers — self-hosted vs cloud

Cecha	Self-hosted	Cloud-managed
RabbitMQ / Kafka	Sam zarządzasz klastrem	Płać za użycie — zero ops
Wydajność	Konfigurowalna	Ilimitowana wertykalnie
SLA	Twoja odpowiedzialność	99.9%+ od dostawcy
Skalowanie	Ręczne / Kafka KRaft	Auto-scaling
Monitoring	Prometheus + Grafana	Wbudowane dashboardy
Koszt	CapEx (serwery) + OpEx	Pay-per-use / ryczałt

Kluczowa decyzja

Mały zespół? Krótkoterminowy projekt? → **Managed broker**.

Potrzebujesz pełnej kontroli lub masz specyficzne wymagania? → **Self-hosted Kafka**.

Managed Message Brokers — serwisy chmurowe

Dostawca	Usługa	Model
AWS	SQS (queue), SNS (pub/sub), EventBridge	Queue + Pub/Sub
Google Cloud	Pub/Sub	Queue + Pub/Sub
Azure	Service Bus, Event Hubs	Queue + Streaming
Confluent	Confluent Cloud (Kafka as a service)	Full Kafka

EventBridge / Pub/Sub

Oprócz queue — **event buses** propagują zdarzenia między kontami, serwisami, nawet partnerami.
EventBridge: reguły → routing bez kodu.

Managed API Gateway — chmurowe rozwiązania

Dostawca	Usługa	Cechy
AWS	API Gateway	REST, HTTP, WebSocket; Lambda integration
Google Cloud	Cloud Endpoints / Gateway	gRPC, REST; Spiffe authentication
Azure	API Management	Policy-based transforms, DevPortal
Kong	Kong Cloud	Plugin ecosystem, multi-cloud

Co dostajesz „za darmo“

- Certyfikaty SSL, darmowe certyfikaty
- Wbudowany rate limiting i throttling
- Integracja z Lambda / Cloud Functions
- Access logs, metryki, tracing — zero konfiguracji

Case study: Chat System

Wymagania

1:1 i grupy, miliony online, dostarczenie < 200ms, persistencja, statusy doręczenia.

Polling / Long-polling

- HTTP co N sekund
- Marnowanie zasobów
- **Nie nadaje się** dla chatu

WebSocket

- Persistent, full-duplex
- Serwer wysyła bez żądania
- Idealne dla real-time

Chat — architektura na skalę

Komponenty

- **API Gateway** — uwierzytelnianie, trasowanie
- **Chat service** — WebSocket, sesje
- **Kafka** — bufor wiadomości, ordering
- **Message store** — Cassandra (write-heavy)
- **Push notification** — dla offline użytkowników

Dlaczego Kafka? Buforowanie szczytu, fan-out grup, replay historii.

System e-commerce. Serwis zamówień musi powiadomić: magazyn, płatności, e-mail.

Synchronicznie (REST) czy asynchronicznie (kolejka)?

Uzasadnij trade-offy.

Podsumowanie

Kluczowe wnioski

1. **REST** — uniwersalne, API publiczne
2. **gRPC** — szybkie, typowane, serwis↔serwis
3. **Message Brokers** — decoupling, odporność, fan-out
4. **Idempotencja** — bez niej retry = katastrofa
5. **Sync vs async** — macierz decyzyjna, nie dogmat
6. **Managed brokers** — wybierz mądrze: SQS/Pub/Sub dla prostoty, Kafka dla kontroli

Źródła i lektury

- M. Kleppmann — *DDIA*, rozdz. 4
- S. Newman — *Building Microservices*, rozdz. 4
- J. Kreps — *The Log* (esej, LinkedIn Engineering)
- A. Xu — *System Design Interview*, rozdz. 12
- AWS Docs — SQS, SNS, EventBridge
- GCP Docs — Cloud Pub/Sub
- Azure Docs — Service Bus, Event Hubs

Następny wykład

Wykład 4: Wzorce architektoniczne

Event-Driven Architecture · CQRS · Event Sourcing · Saga