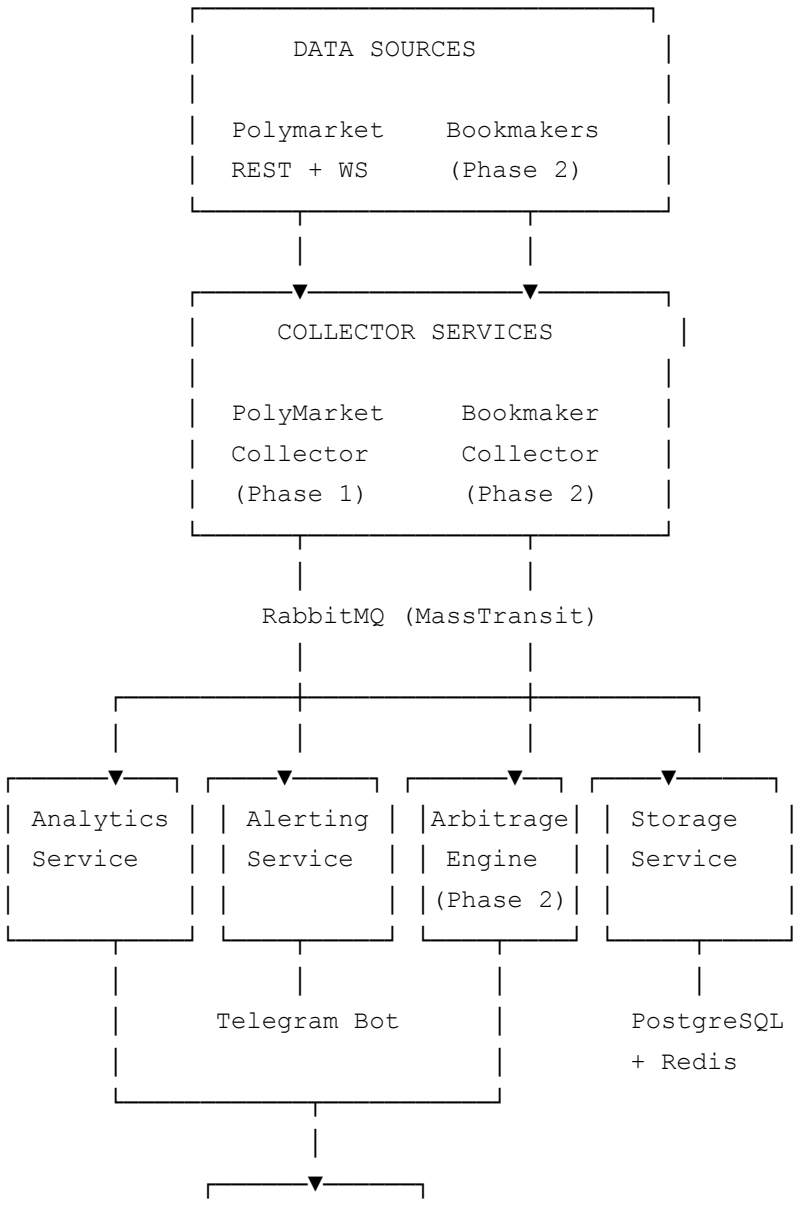
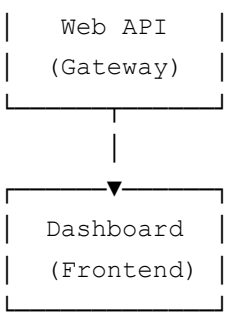
PolyMarket Analytics Platform — Архитектура

Общая идея

Платформа для мониторинга и анализа рынков Polymarket в реальном времени с возможностью расширения модулем арбитража с букмекерскими конторами.

Высокоуровневая архитектура





Сервисы — Phase 1 (Polymarket Analytics)

1. PolyMarket.Collector

Отвечает за сбор данных с Polymarket.

Что делает:

- REST-клиент для Gamma API — загружает список маркетов, метаданные, текущие цены
- WebSocket-клиент для CLOB — подписывается на реал-тайм изменения цен и сделки
- Data API клиент — позиции китов, топ-холдеры, крупные сделки
- Публикует события в RabbitMQ

Ключевые события (Messages):

```
public record MarketPriceChanged(
    string MarketId,
    string Question,
    decimal OldPrice,
    decimal NewPrice,
    decimal ChangePercent,
    DateTime Timestamp);

public record LargeTradeDetected(
    string MarketId,
    string TraderAddress,
    string Side,     // BUY / SELL
    decimal Size,
    decimal Price,
    DateTime Timestamp);
```

```
public record MarketVolumeSpike(
    string MarketId,
    decimal NormalVolume24h,
    decimal CurrentVolume1h,
    decimal SpikeMultiplier,
    DateTime Timestamp);

public record MarketSnapshotUpdated(
    string MarketId,
    string Question,
    decimal YesPrice,
    decimal NoPrice,
    decimal Volume24h,
    decimal Liquidity,
    DateTime Timestamp);
```

Технологии:

- .NET 8, BackgroundService (hosted service)
- System.Net.WebSockets для WS-подключения
- HttpClient + Polly для REST с ретраями
- MassTransit для публикации в RabbitMQ

---

2. PolyMarket.Analytics

Анализирует поток данных и выявляет аномалии.

Что делает:

- Подписывается на события от Collector через MassTransit
- Считает скользящие средние цен и объёмов (по окнам 1h, 6h, 24h)
- Детектит аномалии:
  - Резкое движение цены (>5% за 10 минут)
  - Всплеск объёма (>3x от среднего)
  - Крупные сделки (>$10k)
  - Расхождение связанных маркетов
  - Маркеты близкие к резолву (цена >0.95 или <0.05)

- Генерирует аналитические события (Alert)

Ключевые модели:

```
public record AnomalyDetected(
    AnomalyType Type,
    string MarketId,
    string Description,
    decimal Severity,   // 0.0 - 1.0
    Dictionary<string, object> Details,
    DateTime Timestamp);

public enum AnomalyType
{
    PriceSpike,
    VolumeSpike,
    WhaleTrade,
    MarketDivergence,
    NearResolution,
    ArbitrageOpportunity  // Phase 2
}
```

Технологии:

- .NET 8, MassTransit Consumers
- In-memory кэш (Redis) для скользящих окон
- Математика своя, без тяжёлых ML-библиотек

---

3. PolyMarket.Alerting

Рассылает уведомления.

Что делает:

- Подписывается на AnomalyDetected
- Фильтрует по настройкам пользователя (минимальная severity, типы аномалий, конкретные маркеты)
- Отправляет в Telegram через Bot API
- Хранит историю алертов

Технологии:

- .NET 8, MassTransit Consumer
- Telegram.Bot SDK
- PostgreSQL для хранения подписок и истории

---

4. PolyMarket.Storage

Персистентное хранение всех данных.

Что делает:

- Подписывается на MarketSnapshotUpdated, сохраняет историю цен
- Хранит метаданные маркетов
- Хранит историю сделок и аномалий
- Предоставляет данные для API и аналитики

Схема БД (основные таблицы):

```
-- Маркеты
CREATE TABLE markets (
    id TEXT PRIMARY KEY,           -- Polymarket condition_id
    question TEXT NOT NULL,
    category TEXT,
    end_date TIMESTAMPTZ,
    active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMPTZ DEFAULT NOW()
);

-- История цен (таймсерии)
CREATE TABLE price_history (
    id BIGSERIAL PRIMARY KEY,
    market_id TEXT REFERENCES markets(id),
    yes_price DECIMAL(10,6),
    no_price DECIMAL(10,6),
    volume_24h DECIMAL(18,2),
    liquidity DECIMAL(18,2),
    timestamp TIMESTAMPTZ NOT NULL
);

-- Крупные сделки
CREATE TABLE whale_trades (
    id BIGSERIAL PRIMARY KEY,
    market_id TEXT REFERENCES markets(id),
```

```
    trader_address TEXT,
    side TEXT,
    size DECIMAL(18,6),
    price DECIMAL(10,6),
    timestamp TIMESTAMPTZ NOT NULL
);

-- Аномалии
CREATE TABLE anomalies (
    id BIGSERIAL PRIMARY KEY,
    market_id TEXT REFERENCES markets(id),
    type TEXT NOT NULL,
    severity DECIMAL(3,2),
    description TEXT,
    details JSONB,
    timestamp TIMESTAMPTZ NOT NULL
);

-- Phase 2: Букмекерские коэффициенты
CREATE TABLE bookmaker_odds (
    id BIGSERIAL PRIMARY KEY,
    market_id TEXT,              -- связь с Polymarket маркетом
    bookmaker TEXT NOT NULL,     -- "liga_stavok", "fonbet" etc
    event_name TEXT,
    outcome TEXT,
    odds DECIMAL(10,4),
    implied_probability DECIMAL(10,6),
    timestamp TIMESTAMPTZ NOT NULL
);

-- Phase 2: Арбитражные возможности
CREATE TABLE arbitrage_opportunities (
    id BIGSERIAL PRIMARY KEY,
    polymarket_id TEXT,
    bookmaker TEXT,
    poly_price DECIMAL(10,6),
    bookmaker_odds DECIMAL(10,4),
    poly_implied_prob DECIMAL(10,6),
    bookmaker_implied_prob DECIMAL(10,6),
    spread DECIMAL(10,6),
    potential_profit_pct DECIMAL(10,4),
    detected_at TIMESTAMPTZ NOT NULL,
    closed_at TIMESTAMPTZ           -- когда окно закрылось
);

CREATE INDEX idx_price_history_market_ts ON price_history(market_id, timestamp DE
CREATE INDEX idx_anomalies_market_ts ON anomalies(market_id, timestamp DESC);
```

```
CREATE INDEX idx_whale_trades_market_ts ON whale_trades(market_id, timestamp DESC
```

Технологии:

- .NET 8, EF Core 8
- PostgreSQL (TimescaleDB расширение опционально для таймсерий)
- Redis для кэширования горячих данных

---

5. PolyMarket.WebApi (Gateway)

REST API для дашборда и внешних клиентов.

Эндпоинты:

```
GET  /api/markets                    — список активных маркетов с текущими ценами
GET  /api/markets/{id}               — детали маркета + история цен
GET  /api/markets/{id}/trades        — крупные сделки по маркету
GET  /api/anomalies                  — последние аномалии (фильтр по типу, severi
GET  /api/anomalies/stats            — статистика аномалий за период
GET  /api/whales                     — топ китов и их позиции
GET  /api/whales/{address}/history   — история сделок кита

# Phase 2
GET  /api/arbitrage/opportunities    — текущие арбитражные окна
GET  /api/arbitrage/history          — история арбитражных возможностей
GET  /api/arbitrage/stats            — средний профит, частота окон
```

Технологии:

- .NET 8, Minimal API
- Redis для кэширования ответов
- Swagger / OpenAPI

Phase 2 — Арбитражный модуль

6. Bookmaker.Collector

Сбор коэффициентов с букмекеров.

Что делает:

- Скрапит или тянет API букмекеров (Лига Ставок, Фонбет, 1xBet, Pinnacle)
- Маппит события букмекера на маркеты Polymarket (по названию команд/событий)
- Публикует BookmakerOddsUpdated в RabbitMQ

Важная архитектурная деталь — абстракция источника:

```
// Общий интерфейс для любого источника коэффициентов
public interface IOddsSource
{
    string Name { get; }                    // "polymarket", "liga_stavok", "fonbet
    Task<IReadOnlyList<OddsSnapshot>> GetCurrentOddsAsync(CancellationToken ct);
}

// Единая модель
public record OddsSnapshot(
    string SourceName,
    string EventId,           // внутренний ID события
    string EventName,         // "Real Madrid vs Barcelona"
    string Outcome,           // "Real Madrid Win"
    decimal ImpliedProbability,
    decimal RawOdds,          // коэффициент (для букмекеров) или цена (для Polym
    DateTime Timestamp);
```

Polymarket — это тоже IOddsSource. Благодаря этому арбитражный движок не знает откуда пришли данные — он просто сравнивает вероятности из разных источников.

---

7. Arbitrage.Engine

Что делает:

- Подписывается на обновления цен от всех источников
- Маппит события между источниками (fuzzy matching по названиям)
- Считает implied probability для каждого источника
- Находит расхождения где сумма implied probabilities < 1.0 (арбитраж)
- Генерирует ArbitrageOpportunityDetected

Формула арбитража:

```
Polymarket: "Team A wins" YES = 0.55 → implied prob = 55%
Букмекер:   "Team A loses" коэф 2.30 → implied prob = 43.5%
```

```
Сумма: 55% + 43.5% = 98.5% < 100%
Арбитраж: 100% - 98.5% = 1.5% гарантированный профит
```

Инфраструктура

Docker Compose

```yaml
services:
  polymarket-collector:
    build: ./src/PolyMarket.Collector
    depends_on: [rabbitmq, redis]
    environment:
      - RabbitMQ__Host=rabbitmq
      - Redis__Connection=redis:6379

  analytics:
    build: ./src/PolyMarket.Analytics
    depends_on: [rabbitmq, redis]

  alerting:
    build: ./src/PolyMarket.Alerting
    depends_on: [rabbitmq, postgres]

  storage:
    build: ./src/PolyMarket.Storage
    depends_on: [rabbitmq, postgres]

  webapi:
    build: ./src/PolyMarket.WebApi
    ports: ["5000:8080"]
    depends_on: [postgres, redis]

  # Phase 2
  # bookmaker-collector:
  #   build: ./src/Bookmaker.Collector
  #   depends_on: [rabbitmq, redis]
  #
  # arbitrage-engine:
  #   build: ./src/Arbitrage.Engine
  #   depends_on: [rabbitmq, postgres, redis]

  rabbitmq:
```

```
    image: rabbitmq:3-management
    ports: ["5672:5672", "15672:15672"]

  postgres:
    image: postgres:16
    environment:
      POSTGRES_DB: polymarket
      POSTGRES_USER: app
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    volumes: ["pgdata:/var/lib/postgresql/data"]

  redis:
    image: redis:7-alpine
    ports: ["6379:6379"]

volumes:
  pgdata:
```

Структура Solution

```
PolyMarketAnalytics.sln
│
├── src/
│   ├── PolyMarket.Contracts/          # Shared messages, interfaces, models
│   │   ├── Messages/                  # MassTransit message contracts
│   │   ├── Interfaces/                # IOddsSource и другие абстракции
│   │   └── Models/                    # Domain models
│   │
│   ├── PolyMarket.Collector/          # Background service, WS + REST клиенты
│   │   ├── Clients/
│   │   │   ├── GammaApiClient.cs      # REST — маркеты, метаданные
│   │   │   ├── ClobWebSocketClient.cs # WS — реал-тайм цены и сделки
│   │   │   └── DataApiClient.cs       # REST — позиции, киты
│   │   ├── Workers/
│   │   │   ├── MarketSyncWorker.cs    # Периодическая синхронизация маркетов
│   │   │   └── PriceStreamWorker.cs   # WebSocket стрим
│   │   └── Program.cs
│   │
│   ├── PolyMarket.Analytics/          # Обработка событий, детект аномалий
│   │   ├── Consumers/
│   │   │   ├── PriceChangedConsumer.cs
│   │   │   ├── TradeConsumer.cs
│   │   │   └── VolumeConsumer.cs
│   │   ├── Detectors/
```

```
│   │   │   ├── PriceSpikeDetector.cs
│   │   │   ├── VolumeSpikeDetector.cs
│   │   │   ├── WhaleDetector.cs
│   │   │   └── MarketDivergenceDetector.cs
│   │   └── Program.cs
│   │
│   ├── PolyMarket.Alerting/          # Telegram бот, уведомления
│   │   ├── Consumers/
│   │   │   └── AnomalyAlertConsumer.cs
│   │   ├── Channels/
│   │   │   └── TelegramChannel.cs
│   │   └── Program.cs
│   │
│   ├── PolyMarket.Storage/           # Персистентность
│   │   ├── Consumers/
│   │   │   ├── SnapshotConsumer.cs
│   │   │   ├── TradeConsumer.cs
│   │   │   └── AnomalyConsumer.cs
│   │   ├── Data/
│   │   │   ├── AppDbContext.cs
│   │   │   └── Migrations/
│   │   └── Program.cs
│   │
│   ├── PolyMarket.WebApi/            # Minimal API gateway
│   │   ├── Endpoints/
│   │   │   ├── MarketsEndpoints.cs
│   │   │   ├── AnomaliesEndpoints.cs
│   │   │   └── WhalesEndpoints.cs
│   │   └── Program.cs
│   │
│   ├── # ── Phase 2 ──
│   │
│   ├── Bookmaker.Collector/          # Скрапинг букмекеров
│   │   ├── Sources/
│   │   │   ├── LigaStavokSource.cs
│   │   │   ├── FonbetSource.cs
│   │   │   └── PinnacleSource.cs
│   │   ├── Matching/
│   │   │   └── EventMatcher.cs       # Fuzzy matching событий
│   │   └── Program.cs
│   │
│   └── Arbitrage.Engine/             # Поиск арбитражных окон
│       ├── Consumers/
│       │   └── OddsUpdatedConsumer.cs
│       ├── Calculator/
│       │   └── ArbitrageCalculator.cs
│       └── Program.cs
```

```
│
└── tests/
    ├── PolyMarket.Analytics.Tests/
    ├── Arbitrage.Engine.Tests/
    └── Integration.Tests/
```

План реализации по неделям

Неделя 1-2: Фундамент

- Создать solution, проекты, Docker Compose
- PolyMarket.Contracts — все message contracts и интерфейсы
- Настроить MassTransit + RabbitMQ
- PostgreSQL + EF Core + миграции (все таблицы Phase 1)
- Базовый Collector: REST-клиент для Gamma API (список маркетов, цены)

Неделя 3-4: Реал-тайм данные

- WebSocket клиент для CLOB (стрим цен)
- Data API клиент (киты, крупные сделки)
- Storage Service — сохранение всех событий в БД
- Базовая аналитика: PriceSpikeDetector, VolumeSpikeDetector

Неделя 5-6: Аналитика и алерты

- WhaleDetector, MarketDivergenceDetector
- Redis кэш для скользящих окон
- Alerting Service + Telegram бот
- Web API — базовые эндпоинты

Неделя 7-8: Полировка

- Все эндпоинты API
- Swagger документация
- Docker Compose полностью рабочий в одну команду

- README с описанием архитектуры
- Unit тесты на детекторы и калькуляторы
- Деплой на VPS

Неделя 9-10: Phase 2 (арбитраж)

- Bookmaker.Collector — хотя бы один источник
- EventMatcher — связывание событий
- Arbitrage.Engine — расчёт арбитражных окон
- Дополнительные эндпоинты API
- Алерты на арбитражные возможности

Что говорить на собеседовании

> “Я разработал real-time аналитическую платформу для prediction markets. Микросервисная архитектура на .NET 8 — 6 сервисов, асинхронное взаимодействие через MassTransit + RabbitMQ. Сбор данных через REST API и WebSocket, обработка потока событий, детекция аномалий, алерты в Telegram. PostgreSQL + EF Core для хранения, Redis для кэширования горячих данных. Всё в Docker Compose. Архитектура расширяемая — добавление нового источника данных через реализацию одного интерфейса.”