# Liệt kê các đặc tính chất lượng mong muốn đạt được với Event-driven Architecture trong bài lab04. Liệt kê các công cụ có thể sử dụng và các bước cần thực hiện, để kiểm tra các đặc tính chất lương này. Vẽ sơ đồ lưu trữ của kiến trúc để xuất? Liệt kê các công cụ có thể sử dụng và các bước cần thực hiện để cài đặt sơ đồ lưu trữ đề xuất, và để viết mã nguồn ghi và đọc các sự kiện từ hệ thống lưu trữ.

## 1. Đặc tính Chất lượng Mong muốn với EDA

### A. Scalability (Khả năng mở rộng)

**Định nghĩa**: Hệ thống có thể tăng capacity để xử lý nhiều users/requests hơn

**Cách đạt được trong project Quiz App**:

- **Horizontal scaling**: Mỗi service (User, Quiz, Answer, Scoring) có thể scale độc lập bằng cách tăng số container instances
- **Load distribution**: Kafka partitions phân phối events đến multiple consumer instances
- **Stateless services**: Services không lưu state, dễ dàng replicate
- **Database sharding**: PostgreSQL có thể shard theo quizId hoặc userId

**Metrics đo lường**:

- Concurrent users: từ 100 → 10,000 users
- Events/second: từ 1,000 → 100,000 events/s
- Response time: giữ <200ms khi scale up
- Resource utilization: CPU <80%, Memory <85%

### B. Reliability (Độ tin cậy)

**Định nghĩa**: Hệ thống hoạt động đúng và liên tục, ngay cả khi có failures

**Cách đạt được**:

- **Fault tolerance**: Circuit breaker pattern, service failure không crash toàn hệ thống
- **Event persistence**: Kafka với replication factor=3, đảm bảo events không bị mất
- **Retry mechanisms**: Exponential backoff retry cho failed events
- **Data consistency**: Eventually consistent với compensating transactions
- **Health checks**: Container health monitoring, auto-restart failed services

**Reliability targets**:

- Uptime: 99.9% (8.76 hours downtime/year)

- Data durability: 99.999% (không mất events)
- Recovery time: <30 seconds cho service restart
- MTBF (Mean Time Between Failures): >720 hours

**Benefits**:

- Dễ maintain: Team có thể work độc lập trên mỗi service
- Flexible deployment: Deploy service bất kỳ mà không affect others
- Technology freedom: Chọn tech stack phù hợp cho từng service
- Testing isolation: Unit test service mà không cần dependency

**Performance targets**:

- API response time: <200ms cho 95% requests
- Event processing latency: <50ms từ publish đến consume
- WebSocket message delivery: <20ms
- Cache hit ratio: >90% cho user sessions
- UI responsiveness: <100ms cho user interactions

### C. Availability (Tính khả dụng)

**High Availability design**:

- **Multi-instance deployment**: Mỗi service chạy ≥2 instances
- **Load balancing**: Nginx distribute traffic across instances
- **Database replication**: PostgreSQL master-slave setup
- **Kafka clustering**: 3+ Kafka brokers với leader election
- **Graceful degradation**: Core features vẫn work khi non-critical services down

### D. Consistency (Tính nhất quán)

**Event-driven consistency model**:

- **Eventually consistent**: Data sync across services through events
- **Saga pattern**: Distributed transactions với compensating actions
- **Event sourcing**: Single source of truth từ event sequence
- **CQRS**: Tách read/write models để optimize performance
- **Conflict resolution**: Last-writer-wins với timestamps

## 2. Công cụ Kiểm tra Chất lượng

### A. Performance Testing (Kiểm tra Hiệu suất)

#### A.1 Load Testing Tools

**Công cụ chính**:

- **Apache JMeter**: GUI-based, HTTP/WebSocket testing
- **Artillery**: Node.js based, event-driven load testing
- **k6**: JavaScript-based, modern load testing
- **Gatling**: Scala-based, high-performance testing

**Test scenarios cho Quiz App**:

```
# JMeter test plan
1. Concurrent user registration: 100-1000 users/minute
2. Quiz creation load: 50 quizzes/minute
3. Real-time quiz participation: 500 concurrent players
4. Answer submission spike: 1000 answers trong 10 seconds
5. WebSocket stress test: 2000 concurrent connections
```

**Metrics cần đo**:

- **Response time**: P50, P95, P99 percentiles
- **Throughput**: Requests/second, Events/second
- **Error rate**: <1% cho normal load, <5% cho peak load
- **Resource utilization**: CPU, Memory, Network I/O
- **Database performance**: Connection pool usage, query time

#### A.2 Implementation Steps

```
# Artillery example cho Quiz App
1. Setup test data: users, quizzes, questions
2. Configure load patterns: ramp-up, steady-state, peak
3. Test API endpoints: /auth, /quiz, /answer, /score
4. WebSocket testing: join quiz, receive events
5. Database load: concurrent reads/writes
6. Kafka throughput: event publishing/consuming
```

### B. Reliability Testing (Kiểm tra Độ tin cậy)

#### B.1 Chaos Engineering Tools

**Công cụ**:

- **Chaos Monkey**: Random service termination
- **Chaos Toolkit**: Declarative chaos experiments
- **Litmus**: Kubernetes-native chaos engineering
- **Pumba**: Docker container chaos testing

**Failure scenarios**:

```
# Chaos experiments cho Quiz App
1. Service failures:
   - Kill random microservice instances
   - Simulate OOM (Out of Memory)
   - CPU exhaustion attacks

2. Infrastructure failures:
   - Kafka broker crashes
   - PostgreSQL connection loss
   - Redis cache failures
   - Network partitions

3. Dependency failures:
   - External API timeouts
   - Database query failures
   - Message delivery failures
```

#### B.2 Resilience Testing Steps

1. **Baseline measurement**: Establish normal performance metrics
2. **Chaos injection**: Introduce controlled failures
3. **Recovery monitoring**: Measure recovery time và impact
4. **Failure analysis**: Identify weak points
5. **Improvement iteration**: Fix issues và repeat

**Expected behaviors**:

- **Service recovery**: Auto-restart failed containers trong <30s
- **Data consistency**: No data loss during failures
- **Graceful degradation**: Core functionality vẫn available
- **User experience**: Meaningful error messages, no crashes

#### B.3 Data Integrity Testing

**Tools**: Testcontainers, Docker Compose **Test cases**:

```
1. Database transaction rollback testing
2. Kafka message delivery guarantees
3. Event ordering verification
4. Duplicate event handling
5. Network partition tolerance (CAP theorem)
```

### C. End-to-End Testing (Kiểm tra Tích hợp)

#### C.1 E2E Testing Framework

**Tools**: Cypress, Playwright, Selenium **Test scenarios**:

```
// Cypress E2E test example
1. User journey: Register → Login → Create Quiz → Start Quiz
2. Multi-player scenario: Host starts quiz, 10 players join, answer
questions
3. Real-time updates: Verify leaderboard updates in real-time
```

```
4. Error handling: Network disconnection, service failures
5. Cross-browser compatibility: Chrome, Firefox, Safari
```

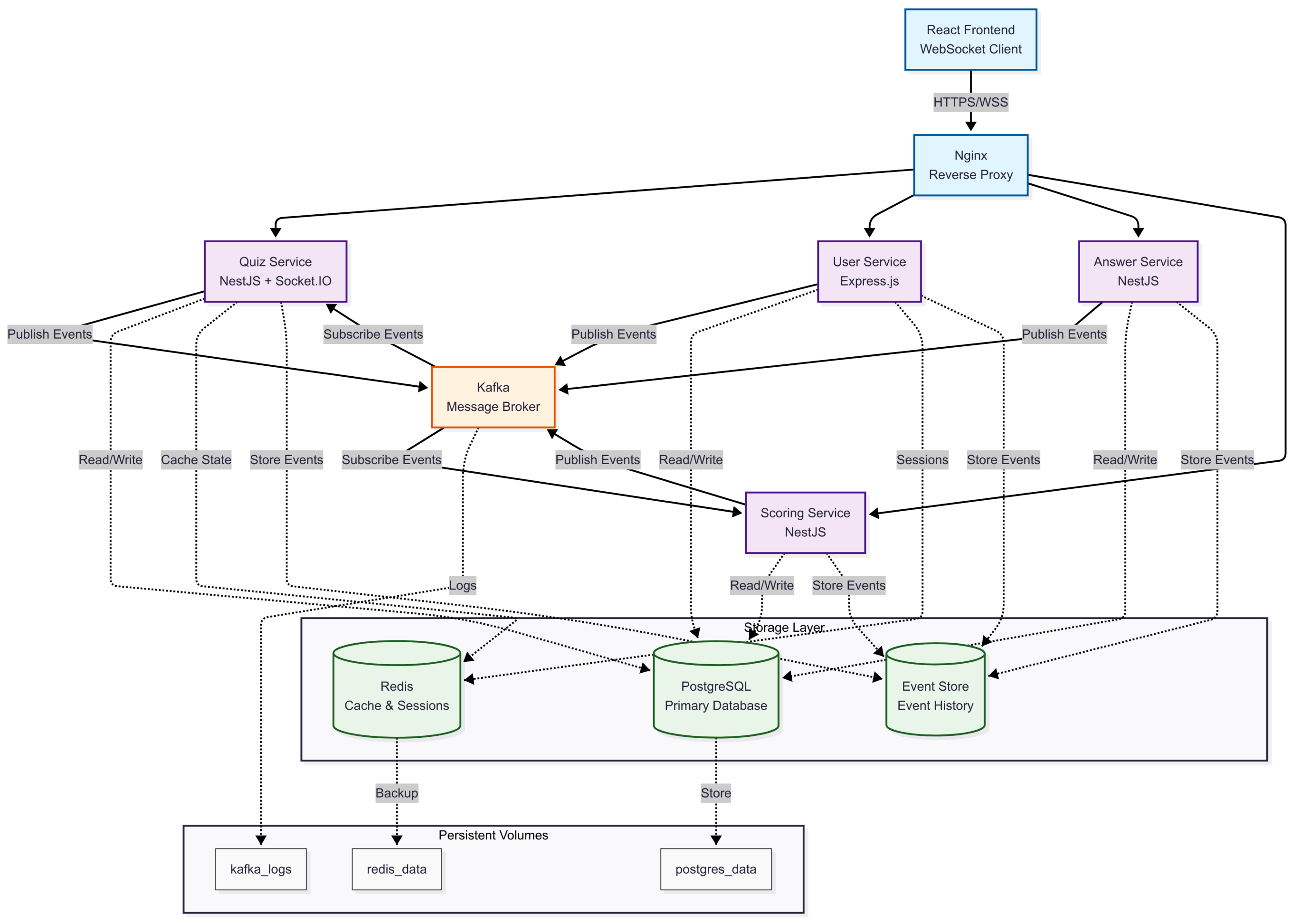
#### C.2 Contract Testing

**Tools**: Pact, Spring Cloud Contract **Purpose**: Verify service integration contracts

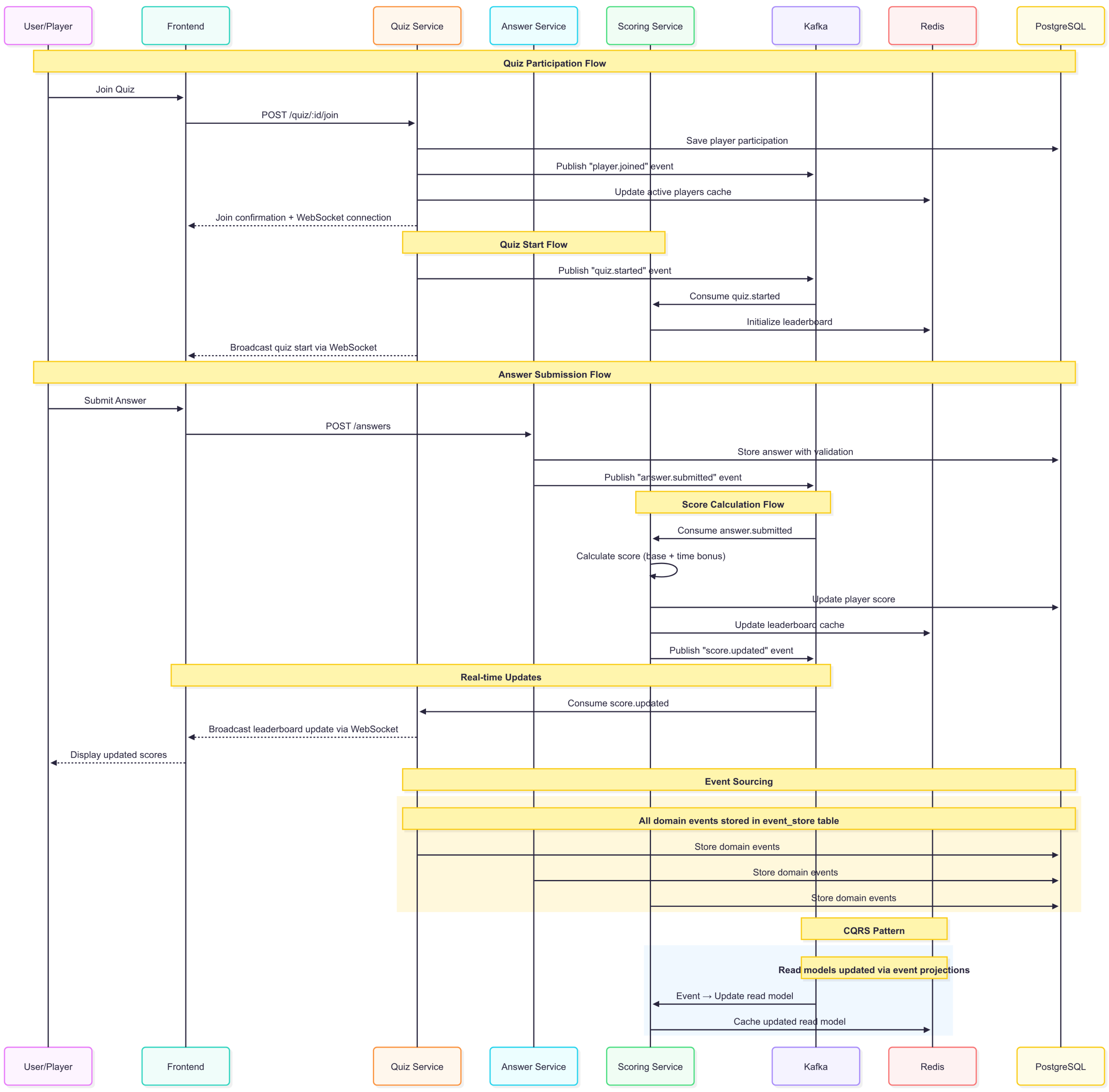
```
# Contract testing approach
1. Define event schemas trong shared/types/
2. Generate contract tests từ schemas
3. Verify producer/consumer compatibility
4. Automate contract validation trong CI/CD
```

## 3. Sơ đồ Kiến trúc Lưu trữ

### 3.1 Tổng quan Storage Architecture



### 3.2 Event Flow Architecture



### 3.5 Data Storage Layers

#### Layer 1: Event Store (Kafka)

**Purpose**: Event sourcing, message streaming **Components**:

- **Topics**: quiz-events, user-events, scoring-events
- **Partitions**: Horizontal scaling (theo quizId)
- **Retention**: 7 days cho event replay
- **Replication**: Factor=3 cho high availability

#### Layer 2: Persistent Storage (PostgreSQL)

**Purpose**: ACID transactions, relational data integrity **Databases**:

- **user_db**: Users, authentication, profiles
- **quiz_db**: Quizzes, questions, quiz participants

- **answer_db**: Answer submissions, validations
- **scoring_db**: Player scores, leaderboards

**Schema design**:

```sql
-- Event Store table (cross-service)
CREATE TABLE event_store (
  id UUID PRIMARY KEY,
  aggregate_id UUID NOT NULL,
  event_type VARCHAR(100) NOT NULL,
  event_data JSONB NOT NULL,
  version INTEGER NOT NULL,
  created_at TIMESTAMP DEFAULT NOW(),
  INDEX (aggregate_id, version)
);
```

**Layer 3: Cache Layer (Redis)**

**Purpose**: Fast access, session management, temporary state **Data types**:

- **String**: User sessions, JWT blacklist
- **Hash**: Quiz state, player current status
- **List**: Question queues, answer history
- **Set**: Active participants, online users
- **Sorted Set**: Leaderboards, rankings
- **TTL**: Auto-expiration cho temporary data

**Layer 4: File Storage**

**Purpose**: Static assets, uploads, exports **Solutions**:

- **Local volumes**: Development environment
- **AWS S3/MinIO**: Production file storage
- **CDN**: Global content delivery

## 4. Cài đặt Kiến trúc Lưu trữ

### 4.1 Infrastructure Setup với Docker Compose

**A. Kafka Cluster Setup**

**Công cụ**: Docker Compose, Confluent Platform, Kafdrop UI

**Configuration trong docker-compose.yml**:

```yaml
# Zookeeper cho Kafka coordination
zookeeper:
  image: confluentinc/cp-zookeeper:7.0.1
  environment:
    ZOOKEEPER_CLIENT_PORT: 2181
    ZOOKEEPER_TICK_TIME: 2000
  volumes:
    - zookeeper-data:/var/lib/zookeeper/data

# Kafka broker với production-ready config
kafka:
  image: confluentinc/cp-kafka:7.0.1
  depends_on: [zookeeper]
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:9092
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    KAFKA_AUTO_CREATE_TOPICS_ENABLE: true
    KAFKA_NUM_PARTITIONS: 3
  volumes:
    - kafka-data:/var/lib/kafka/data
```

**Bước setup chi tiết**:

```
# 1. Start Kafka infrastructure
docker-compose up -d zookeeper kafka

# 2. Wait for Kafka to be ready
docker-compose exec kafka kafka-topics --bootstrap-server localhost:9092 --list

# 3. Create topics với specific configuration
docker-compose exec kafka kafka-topics --create \
  --bootstrap-server localhost:9092 \
  --topic quiz-events \
  --partitions 6 \
  --replication-factor 1 \
  --config retention.ms=604800000

# 4. Create consumer groups
docker-compose exec kafka kafka-consumer-groups \
  --bootstrap-server localhost:9092 \
  --list

# 5. Monitor topic health
docker-compose exec kafka kafka-log-dirs \
  --bootstrap-server localhost:9092 \
  --describe --json
```

**Topic design cho Quiz App**:

```
Topics:
  quiz-events:
    partitions: 6 (theo quizId hash)
    retention: 7 days
    cleanup.policy: delete

  user-events:
    partitions: 3 (theo userId hash)
    retention: 30 days
    cleanup.policy: compact

  scoring-events:
    partitions: 6 (theo quizId hash)
    retention: 14 days
    cleanup.policy: delete
```

#### B. PostgreSQL Multi-Database Setup

**Công cụ**: Docker, pgAdmin, Database migration tools

**Configuration**:

```
# Primary PostgreSQL instance
postgres:
  image: postgres:15
  environment:
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: password
    POSTGRES_MULTIPLE_DATABASES: user_db,quiz_db,answer_db,scoring_db
  volumes:
    - postgres-data:/var/lib/postgresql/data
    - ./infra/postgres/init:/docker-entrypoint-initdb.d
  ports:
    - "5432:5432"
```

**Database initialization scripts**:

```
-- 01-init.sql: Create multiple databases
CREATE DATABASE user_db;
CREATE DATABASE quiz_db;
CREATE DATABASE answer_db;
CREATE DATABASE scoring_db;

-- 02-quiz-schema.sql
\c quiz_db;
CREATE TABLE quizzes (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  title VARCHAR(255) NOT NULL,
  description TEXT,
  host_id UUID NOT NULL,
  status VARCHAR(50) DEFAULT 'draft',
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE questions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  quiz_id UUID REFERENCES quizzes(id) ON DELETE CASCADE,
  question_text TEXT NOT NULL,
  question_type VARCHAR(50) DEFAULT 'multiple_choice',
  correct_answer JSONB NOT NULL,
  options JSONB,
  points INTEGER DEFAULT 10,
  time_limit INTEGER DEFAULT 30,
  order_index INTEGER NOT NULL
);

-- 03-answers-schema.sql
\c answer_db;
CREATE TABLE answers (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  quiz_id UUID NOT NULL,
  question_id UUID NOT NULL,
  player_id UUID NOT NULL,
  submitted_answer JSONB NOT NULL,
  is_correct BOOLEAN NOT NULL,
  response_time INTEGER NOT NULL,
  submitted_at TIMESTAMP DEFAULT NOW(),
  UNIQUE(quiz_id, question_id, player_id)
);

-- 04-event-store.sql: Cross-service event store
CREATE TABLE event_store (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  aggregate_id UUID NOT NULL,
  aggregate_type VARCHAR(100) NOT NULL,
  event_type VARCHAR(100) NOT NULL,
  event_data JSONB NOT NULL,
  event_version INTEGER NOT NULL,
  sequence_number BIGSERIAL,
  created_at TIMESTAMP DEFAULT NOW(),
  UNIQUE(aggregate_id, event_version)
);

CREATE INDEX idx_event_store_aggregate ON event_store(aggregate_id,
event_version);
CREATE INDEX idx_event_store_type ON event_store(event_type);
CREATE INDEX idx_event_store_created ON event_store(created_at);
```

**Connection pooling setup**:

```
# pgbouncer cho connection pooling
pgbouncer:
  image: pgbouncer/pgbouncer:latest
  environment:
    DATABASES_HOST: postgres
    DATABASES_PORT: 5432
    DATABASES_USER: postgres
    DATABASES_PASSWORD: password
    POOL_MODE: transaction
    SERVER_RESET_QUERY: DISCARD ALL
    MAX_CLIENT_CONN: 100
    DEFAULT_POOL_SIZE: 20
```

#### C. Redis Cluster Setup

**Công cụ**: Redis Docker, Redis Sentinel, Redis Cluster

**Single-node development setup**:

```
redis:
  image: redis:7-alpine
  command: redis-server --appendonly yes --replica-read-only no
  volumes:
    - redis-data:/data
  ports:
    - "6379:6379"
```

**Production cluster setup**:

```
# Redis cluster với 3 masters + 3 replicas
redis-cluster:
  image: redis:7-alpine
  command: redis-cli --cluster create \
    redis-1:6379 redis-2:6379 redis-3:6379 \
    redis-4:6379 redis-5:6379 redis-6:6379 \
    --cluster-replicas 1 --cluster-yes
```

**Redis configuration cho Quiz App**:

```
# redis.conf
maxmemory 256mb
maxmemory-policy allkeys-lru
save 900 1
save 300 10
save 60 10000
```

```
appendonly yes
appendfsync everysec
```

### 4.2 Development Environment Setup

#### A. Local Development với Docker Compose

```
# 1. Clone repository
git clone <quiz-app-repo>
cd quiz-app-event-driven

# 2. Start infrastructure services
docker-compose up -d postgres redis kafka zookeeper

# 3. Wait for services to be ready
./scripts/wait-for-services.sh

# 4. Run database migrations
npm run migrate:dev

# 5. Start microservices trong development mode
npm run dev:user-service &
npm run dev:quiz-service &
npm run dev:answer-service &
npm run dev:scoring-service &

# 6. Start frontend
cd frontend && npm start
```

#### B. Production Deployment

**Orchestration**: Docker Swarm hoặc Kubernetes

**Docker Swarm setup**:

```
# docker-stack.yml
version: '3.8'
services:
  quiz-service:
    image: quiz-app/quiz-service:latest
    deploy:
      replicas: 3
      update_config:
        parallelism: 1
        delay: 30s
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
    environment:
      NODE_ENV: production
      DATABASE_URL: postgresql://postgres:password@postgres:5432/quiz_db
      KAFKA_BROKERS: kafka:9092
      REDIS_URL: redis://redis:6379
```

**Kubernetes deployment**:

```yaml
# k8s-quiz-service.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: quiz-service
spec:
  replicas: 3
  selector:
    matchLabels:
      app: quiz-service
  template:
    metadata:
      labels:
        app: quiz-service
    spec:
      containers:
      - name: quiz-service
        image: quiz-app/quiz-service:latest
        ports:
        - containerPort: 3000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: db-secret
              key: url
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
```

## 5. Code Implementation Chi tiết

### 5.1 Event Sourcing & CQRS Implementation

**A. Event Store Repository**

```
// shared/repositories/event-store.repository.ts
@Injectable()
export class EventStoreRepository {
  constructor(
    @InjectRepository(EventStoreEntity)
    private eventStoreRepo: Repository<EventStoreEntity>
  ) {}

  async saveEvent(aggregateId: string, aggregateType: string,
                 eventType: string, eventData: any, version: number):
Promise<void> {
    const event = this.eventStoreRepo.create({
      aggregateId,
      aggregateType,
      eventType,
      eventData,
      eventVersion: version,
      createdAt: new Date()
    });

    try {
      await this.eventStoreRepo.save(event);
      // Publish to Kafka cho real-time processing
      await this.kafkaService.publishEvent(event);
    } catch (error) {
      if (error.code === '23505') { // Unique constraint violation
        throw new ConflictException(`Optimistic concurrency violation for
${aggregateId}`);
      }
      throw error;
    }
  }

  async getEvents(aggregateId: string, fromVersion?: number):
Promise<EventStoreEntity[]> {
    const query = this.eventStoreRepo
      .createQueryBuilder('event')
      .where('event.aggregateId = :aggregateId', { aggregateId })
      .orderBy('event.eventVersion', 'ASC');

    if (fromVersion) {
      query.andWhere('event.eventVersion > :fromVersion', { fromVersion
});
    }

    return await query.getMany();
  }

  async getEventsByType(eventType: string, fromDate: Date):
Promise<EventStoreEntity[]> {
    return await this.eventStoreRepo.find({
      where: {
        eventType,
        createdAt: MoreThan(fromDate)
      },
      order: { createdAt: 'ASC' }
    });
  }
}
```

#### B. Aggregate Root Base Class

```
// shared/domain/aggregate-root.ts
export abstract class AggregateRoot {
  protected id: string;
  protected version: number = 0;
  private uncommittedEvents: DomainEvent[] = [];

  constructor(id: string) {
    this.id = id;
  }

  protected addEvent(event: DomainEvent): void {
    this.uncommittedEvents.push(event);
    this.version += 1;
  }

  public getUncommittedEvents(): DomainEvent[] {
    return [...this.uncommittedEvents];
  }

  public clearEvents(): void {
    this.uncommittedEvents = [];
  }

  public static fromEvents<T extends AggregateRoot>(
    constructor: new (id: string) => T,
    events: DomainEvent[]
  ): T {
    const aggregate = new constructor(events[0].aggregateId);

    for (const event of events) {
      aggregate.applyEvent(event, false);
    }

    aggregate.clearEvents();
    return aggregate;
  }

  protected abstract applyEvent(event: DomainEvent, isNew: boolean): void;
}
```

#### C. Quiz Aggregate Implementation

```
// services/quiz-service/src/domain/quiz.aggregate.ts
export class QuizAggregate extends AggregateRoot {
  private title: string;
  private questions: Question[] = [];
  private participants: string[] = [];
  private status: QuizStatus = QuizStatus.DRAFT;
  private currentQuestionIndex: number = 0;

  constructor(id: string) {
    super(id);
  }

  public createQuiz(title: string, hostId: string, questions: Question[]):
void {
    if (this.version > 0) {
      throw new Error('Quiz already exists');
    }

    const event = new QuizCreatedEvent(this.id, title, hostId, questions);
    this.addEvent(event);
    this.applyEvent(event, true);
  }

  public startQuiz(): void {
    if (this.status !== QuizStatus.READY) {
      throw new Error('Quiz must be in READY status to start');
    }

    const event = new QuizStartedEvent(this.id, this.participants);
    this.addEvent(event);
    this.applyEvent(event, true);
  }

  public addParticipant(playerId: string): void {
    if (this.participants.includes(playerId)) {
      throw new Error('Player already joined');
    }

    const event = new PlayerJoinedEvent(this.id, playerId);
    this.addEvent(event);
    this.applyEvent(event, true);
  }

  protected applyEvent(event: DomainEvent, isNew: boolean): void {
    switch (event.eventType) {
      case 'quiz.created':
        this.applyQuizCreated(event as QuizCreatedEvent);
        break;
      case 'quiz.started':
        this.applyQuizStarted(event as QuizStartedEvent);
        break;
      case 'player.joined':
        this.applyPlayerJoined(event as PlayerJoinedEvent);
        break;
      default:
        throw new Error(`Unknown event type: ${event.eventType}`);
    }

    if (!isNew) {
      this.version += 1;
    }
  }

  private applyQuizCreated(event: QuizCreatedEvent): void {
    this.title = event.title;
    this.questions = event.questions;
    this.status = QuizStatus.DRAFT;
  }

  private applyQuizStarted(event: QuizStartedEvent): void {
    this.status = QuizStatus.ACTIVE;
    this.currentQuestionIndex = 0;
  }

  private applyPlayerJoined(event: PlayerJoinedEvent): void {
    this.participants.push(event.playerId);
  }
}
```

### 5.2 Event Publishing & Consuming

#### A. Advanced Kafka Producer với Retry Logic

```
// shared/services/kafka-producer.service.ts
@Injectable()
export class KafkaProducerService {
  private producer: Producer;
  private readonly retryConfig: RetryConfig = {
    maxRetries: 3,
    initialRetryDelay: 1000,
    maxRetryDelay: 5000,
    retryMultiplier: 2
  };

  async onModuleInit() {
    this.producer = this.kafka.producer({
      groupId: 'quiz-app-producers',
      transactionTimeout: 30000,
      retry: {
        initialRetryTime: 100,
        retries: 8
      }
    });
```

```
      await this.producer.connect();
    }

    async publishEvent(event: DomainEvent): Promise<void> {
      const message = {
        key: event.aggregateId,
        value: JSON.stringify({
          eventId: event.eventId,
          eventType: event.eventType,
          aggregateId: event.aggregateId,
          aggregateType: event.aggregateType,
          payload: event.payload,
          version: event.version,
          timestamp: event.timestamp
        }),
        headers: {
          'event-type': event.eventType,
          'aggregate-type': event.aggregateType,
          'content-type': 'application/json'
        }
      };

      const topic = this.getTopicForEvent(event.eventType);

      await this.retryWrapper(async () => {
        await this.producer.send({
          topic,
          messages: [message]
        });
      });

      this.logger.log(`Event published: ${event.eventType} for
  ${event.aggregateId}`);
    }

    async publishBatch(events: DomainEvent[]): Promise<void> {
      const groupedByTopic = events.reduce((acc, event) => {
        const topic = this.getTopicForEvent(event.eventType);
        if (!acc[topic]) acc[topic] = [];

        acc[topic].push({
          key: event.aggregateId,
          value: JSON.stringify(event),
          headers: {
            'event-type': event.eventType,
            'aggregate-type': event.aggregateType
          }
        });

        return acc;
      }, {} as Record<string, any[]>);

      const topicMessages = Object.entries(groupedByTopic).map(([topic,
  messages]) => ({
        topic,
        messages
      }));

      await this.retryWrapper(async () => {
        await this.producer.sendBatch({ topicMessages });
      });
    }

    private async retryWrapper<T>(operation: () => Promise<T>): Promise<T> {
      let lastError: Error;

      for (let attempt = 0; attempt <= this.retryConfig.maxRetries;
attempt++) {
        try {
          return await operation();
        } catch (error) {
          lastError = error;

          if (attempt === this.retryConfig.maxRetries) {
            break;
          }

          const delay = Math.min(
            this.retryConfig.initialRetryDelay *
Math.pow(this.retryConfig.retryMultiplier, attempt),
            this.retryConfig.maxRetryDelay
          );

          this.logger.warn(`Retry attempt ${attempt + 1} after ${delay}ms:
${error.message}`);
          await new Promise(resolve => setTimeout(resolve, delay));
        }
      }

      throw new Error(`Failed after ${this.retryConfig.maxRetries} retries:
${lastError.message}`);
    }

    private getTopicForEvent(eventType: string): string {
      const topicMap: Record<string, string> = {
        'quiz.created': 'quiz-events',
        'quiz.started': 'quiz-events',
        'quiz.ended': 'quiz-events',
        'player.joined': 'user-events',
        'answer.submitted': 'answer-events',
        'score.updated': 'scoring-events'
      };

      return topicMap[eventType] || 'default-events';
    }
  }
```

### B. Event Consumer với Error Handling

```
// services/scoring-service/src/services/event-consumer.service.ts
@Injectable()
export class ScoringEventConsumerService {
  constructor(
    private scoringService: ScoringService,
    private deadLetterService: DeadLetterService,
    private metricsService: MetricsService
  ) {}

  @EventPattern('answer.submitted')
  async handleAnswerSubmitted(context: KafkaContext): Promise<void> {
    const message = context.getMessage();
    const event = JSON.parse(message.value.toString()) as
AnswerSubmittedEvent;

    const startTime = Date.now();

    try {
      await this.processAnswerSubmitted(event);

      // Metrics collection
      this.metricsService.recordEventProcessingTime(
        'answer.submitted',
        Date.now() - startTime
      );

this.metricsService.incrementEventProcessedCounter('answer.submitted',
'success');

    } catch (error) {
      this.logger.error(`Failed to process answer.submitted event:
${error.message}`, {
        eventId: event.eventId,
        aggregateId: event.aggregateId,
        error: error.stack
      });

      await this.handleProcessingError(event, error, context);
    }
  }

  private async processAnswerSubmitted(event: AnswerSubmittedEvent):
Promise<void> {
    // Idempotency check
    const existingProcessing = await
this.checkProcessedEvent(event.eventId);
    if (existingProcessing) {
      this.logger.info(`Event ${event.eventId} already processed,
skipping`);
      return;
    }

    // Business logic
    const score = await this.scoringService.calculateScore({
      quizId: event.aggregateId,
      playerId: event.payload.playerId,
      questionId: event.payload.questionId,
      isCorrect: event.payload.isCorrect,
      responseTime: event.payload.responseTime,
      submittedAt: new Date(event.timestamp)
    });

    await this.scoringService.updatePlayerScore(score);

    // Mark as processed
    await this.markEventProcessed(event.eventId);

    // Publish derived events
    if (score.isNewLeader) {
      await this.kafkaProducer.publishEvent(
        new LeaderboardUpdatedEvent(event.aggregateId, score.playerId,
score.totalScore)
      );
    }
  }

  private async handleProcessingError(
    event: DomainEvent,
    error: Error,
    context: KafkaContext
  ): Promise<void> {
    const retryCount = this.getRetryCount(context);
    const maxRetries = 3;

    if (retryCount < maxRetries) {
      // Schedule retry với exponential backoff
      const delay = Math.pow(2, retryCount) * 1000; // 1s, 2s, 4s

      setTimeout(async () => {
        try {
          await this.processAnswerSubmitted(event as
AnswerSubmittedEvent);
        } catch (retryError) {
          await this.handleProcessingError(event, retryError, context);
        }
      }, delay);

    } else {
      // Send to dead letter queue
      await this.deadLetterService.sendToDeadLetter({
        originalEvent: event,
        error: error.message,
        retryCount,
        failedAt: new Date()
      });


this.metricsService.incrementEventProcessedCounter('answer.submitted',
'failed');
    }
  }

  private getRetryCount(context: KafkaContext): number {
    const headers = context.getMessage().headers;
    return parseInt(headers?.['retry-count']?.toString() || '0');
  }
}
```

### 5.3 CQRS Read Models & Projections

#### A. Read Model cho Leaderboard

```
// services/scoring-service/src/projections/leaderboard.projection.ts
@Injectable()
export class LeaderboardProjection {
  constructor(
    @InjectRepository(LeaderboardReadModel)
    private leaderboardRepo: Repository<LeaderboardReadModel>,
    private redisService: RedisService
  ) {}

  @EventPattern('score.updated')
  async onScoreUpdated(event: ScoreUpdatedEvent): Promise<void> {
    const { quizId, playerId, totalScore, rank } = event.payload;

    // Update persistent read model
    await this.leaderboardRepo.upsert({
      quizId,
      playerId,
      totalScore,
      rank,
      lastUpdated: new Date(event.timestamp)
    }, ['quizId', 'playerId']);

    // Update Redis cache cho real-time access
    await this.updateRedisLeaderboard(quizId, playerId, totalScore);

    // Notify via WebSocket
    await this.notifyLeaderboardUpdate(quizId, {
      playerId,
      totalScore,
      rank
    });
  }
  private async updateRedisLeaderboard(quizId: string, playerId: string,
score: number): Promise<void> {
    const key = `leaderboard:${quizId}`;

    // Use Redis Sorted Set cho efficient ranking
    await this.redisService.zadd(key, score, playerId);
    await this.redisService.expire(key, 3600); // 1 hour TTL
  }

  async getLeaderboard(quizId: string, limit: number = 10):
Promise<LeaderboardEntry[]> {
    const cacheKey = `leaderboard:${quizId}`;

    // Try Redis first
    const cachedData = await this.redisService.zrevrange(
      cacheKey, 0, limit - 1, 'WITHSCORES'
    );

    if (cachedData.length > 0) {
      return this.formatLeaderboardData(cachedData);
    }

    // Fallback to database
    const dbData = await this.leaderboardRepo.find({
      where: { quizId },
      order: { totalScore: 'DESC', lastUpdated: 'ASC' },
      take: limit
    });

    // Populate cache
    if (dbData.length > 0) {
      const pipeline = this.redisService.pipeline();
      dbData.forEach(entry => {
        pipeline.zadd(cacheKey, entry.totalScore, entry.playerId);
      });
      pipeline.expire(cacheKey, 3600);
      await pipeline.exec();
    }

    return dbData.map(entry => ({
      playerId: entry.playerId,
      totalScore: entry.totalScore,
      rank: entry.rank
    }));
  }
}
```

#### B. Event Handler với Saga Pattern

```
// services/quiz-service/src/sagas/quiz-completion.saga.ts
@Injectable()
```

```
export class QuizCompletionSaga {
  private sagaStates = new Map<string, QuizCompletionState>();

  @EventPattern('quiz.ended')
  async onQuizEnded(event: QuizEndedEvent): Promise<void> {
    const sagaId = `quiz-completion-${event.aggregateId}`;
    const state = new QuizCompletionState(sagaId, event.aggregateId);

    this.sagaStates.set(sagaId, state);

    // Step 1: Finalize all scores
    await this.finalizeScores(event.aggregateId);
    state.markScoresFinalized();

    // Step 2: Generate quiz report
    await this.generateQuizReport(event.aggregateId);
    state.markReportGenerated();

    // Step 3: Notify participants
    await this.notifyParticipants(event.aggregateId,
event.payload.participants);
    state.markParticipantsNotified();

    // Complete saga
    state.markCompleted();
    this.sagaStates.delete(sagaId);
  }

  @EventPattern('score.finalization.failed')
  async onScoreFinalizationFailed(event: ScoreFinalizationFailedEvent):
Promise<void> {
    const sagaId = `quiz-completion-${event.aggregateId}`;
    const state = this.sagaStates.get(sagaId);

    if (state) {
      // Compensating action: revert quiz to ended-with-errors state
      await this.revertQuizCompletion(event.aggregateId);
      state.markFailed('Score finalization failed');
    }
  }

  private async finalizeScores(quizId: string): Promise<void> {
    try {
      await this.kafkaProducer.publishEvent(
        new FinalizeScoresCommand(quizId)
      );
    } catch (error) {
      await this.kafkaProducer.publishEvent(
        new ScoreFinalizationFailedEvent(quizId, error.message)
      );
      throw error;
    }
  }
  private async generateQuizReport(quizId: string): Promise<void> {
    // Generate comprehensive quiz analytics
    const analytics = await
this.analyticsService.generateQuizReport(quizId);

    await this.kafkaProducer.publishEvent(
      new QuizReportGeneratedEvent(quizId, analytics)
    );
  }

  private async notifyParticipants(quizId: string, participants:
string[]): Promise<void> {
    const notifications = participants.map(playerId =>
      new ParticipantNotificationEvent(quizId, playerId, 'quiz_completed')
    );

    await this.kafkaProducer.publishBatch(notifications);
  }
}
```

## Vẽ và giải thích góc nhìn triển khai của Event-Driven Architecture được đề xuất trong bài lab 04? Liệt kê các công cụ có thể sử dụng và các bước cần thực hiện để triển khai hệ thống theo góc nhìn đề xuất.

1. Góc nhìn Triển khai (Deployment View) của Event-Driven Architecture

**1.1 Tổng quan Deployment Architecture**

Góc nhìn triển khai mô tả cách các thành phần phần mềm được phân phối và triển khai trên hạ tầng vật lý/ảo hoá, bao gồm servers, containers, networks và dependencies.

**Production Deployment Architecture:**

```
graph TB
    subgraph "Frontend Layer"
        WEB[React App
Load Balanced]
    end

    subgraph "API Gateway"
        LB[Nginx Load Balancer
:80, :443]
    end

    subgraph "Microservices"
        US[User Service
:3001]
        QS[Quiz Service
:3000]
        AS[Answer Service
:3002]
```

```
        SS[Scoring Service
:3003]
    end

    subgraph "Event Streaming"
        KAFKA[Kafka Cluster
:9092]
        ZK[Zookeeper
:2181]
    end

    subgraph "Data Storage"
        PG[(PostgreSQL
:5432)]
        REDIS[(Redis
:6379)]
    end

    subgraph "Monitoring"
        MON[Kafdrop & pgAdmin]
    end

    %% Main flow
    WEB --> LB
    LB --> US
    LB --> QS
    LB --> AS
    LB --> SS

    %% Event connections
    US -.->|events| KAFKA
    QS -.->|events| KAFKA
    AS -.->|events| KAFKA
    SS -.->|events| KAFKA

    %% Data connections
    US --> PG
    QS --> PG
    QS --> REDIS
    AS --> PG
    SS --> PG

    %% Infrastructure
    KAFKA --> ZK
    MON --> KAFKA
    MON --> PG

    classDef frontend fill:#e8f5e8,stroke:#2e7d32,stroke-width:2px
    classDef service fill:#e1f5fe,stroke:#01579b,stroke-width:2px
    classDef data fill:#f3e5f5,stroke:#4a148c,stroke-width:2px
    classDef event fill:#fff3e0,stroke:#e65100,stroke-width:2px
    classDef infra fill:#f5f5f5,stroke:#424242,stroke-width:2px

    class WEB frontend
    class LB infra
    class US,QS,AS,SS service
    class PG,REDIS data
    class KAFKA,ZK event
    class MON infra
```

### 1.2 Container Deployment với Docker Swarm

**Docker Swarm Stack Architecture:**

```
graph TB
    subgraph "Docker Swarm Cluster"
        subgraph "Manager Nodes"
            M1[Manager Node 1
quiz-app-manager-1]
            M2[Manager Node 2
quiz-app-manager-2]
            M3[Manager Node 3
quiz-app-manager-3]
        end

        subgraph "Worker Nodes"
            W1[Worker Node 1
quiz-app-worker-1]
            W2[Worker Node 2
quiz-app-worker-2]
            W3[Worker Node 3
quiz-app-worker-3]
            W4[Worker Node 4
quiz-app-worker-4]
        end
    end

    subgraph "Service Distribution"
        subgraph "Frontend Services"
            FS[frontend-service
replicas: 2
ports: 3000:3000]
        end

        subgraph "Backend Services"
            USS[user-service
replicas: 2
ports: 3001:3001]
            QSS[quiz-service
replicas: 3
ports: 3000:3000]
            ASS[answer-service
replicas: 2
ports: 3002:3002]
            SSS[scoring-service
replicas: 2
ports: 3003:3003]
        end

        subgraph "Infrastructure Services"
            PGSVC[postgres-service
replicas: 1
ports: 5432:5432]
            RDSVC[redis-service
replicas: 1
ports: 6379:6379]
            KFSVC[kafka-service
replicas: 3
ports: 9092:9092]
            ZKSVC[zookeeper-service
replicas: 3
ports: 2181:2181]
        end

        subgraph "Proxy Services"
            NGSVC[nginx-service
replicas: 2
ports: 80:80,443:443]
        end
    end

    %% Node assignments
    M1 --> NGSVC
    M2 --> PGSVC
    M3 --> KFSVC

    W1 --> FS
    W1 --> USS
    W2 --> QSS
    W2 --> ASS
    W3 --> SSS
    W3 --> RDSVC
    W4 --> ZKSVC

    classDef manager fill:#ffebee,stroke:#c62828,stroke-width:2px
    classDef worker fill:#e8f5e8,stroke:#2e7d32,stroke-width:2px
    classDef service fill:#e1f5fe,stroke:#01579b,stroke-width:2px

    class M1,M2,M3 manager
    class W1,W2,W3,W4 worker
    class FS,USS,QSS,ASS,SSS,PGSVC,RDSVC,KFSVC,ZKSVC,NGSVC service
```

### 1.3 Kubernetes Deployment Architecture

**Kubernetes Cluster Layout:**

```
graph TB
    subgraph "Kubernetes Cluster"
        subgraph "Control Plane"
            API[API Server]
            ETCD[etcd]
            SCHED[Scheduler]
            CTRL[Controller Manager]
        end

        subgraph "quiz-app Namespace"
            subgraph "Frontend Deployment"
                FPOD1[frontend-pod-1]
                FPOD2[frontend-pod-2]
                FSVC[frontend-service
LoadBalancer]
            end

            subgraph "Backend Deployments"
                UPOD1[user-pod-1]
                UPOD2[user-pod-2]
                USVC[user-service
ClusterIP]

                QPOD1[quiz-pod-1]
                QPOD2[quiz-pod-2]
                QPOD3[quiz-pod-3]
                QSVC[quiz-service
ClusterIP]

                APOD1[answer-pod-1]
                APOD2[answer-pod-2]
                ASVC[answer-service
ClusterIP]

                SPOD1[scoring-pod-1]
                SPOD2[scoring-pod-2]
                SSVC[scoring-service
ClusterIP]
            end

            subgraph "Data StatefulSets"
                PGSTS[postgres-statefulset
replicas: 2]
                RDSTS[redis-statefulset
replicas: 2]
                KFSTS[kafka-statefulset
replicas: 3]
                ZKSTS[zookeeper-statefulset
replicas: 3]
            end

            subgraph "Storage"
                PV1[postgres-pv-1]
                PV2[postgres-pv-2]
                RV1[redis-pv-1]
                KV1[kafka-pv-1]
                KV2[kafka-pv-2]
                KV3[kafka-pv-3]
            end

            subgraph "Configuration"
                CM[ConfigMaps]
                SEC[Secrets]
            end
        end

        subgraph "Ingress"
            ING[nginx-ingress
*.quiz-app.com]
        end
    end

    %% Service connections
    FSVC --> FPOD1
    FSVC --> FPOD2
    USVC --> UPOD1
    USVC --> UPOD2
    QSVC --> QPOD1
    QSVC --> QPOD2
    QSVC --> QPOD3
    ASVC --> APOD1
    ASVC --> APOD2
    SSVC --> SPOD1
    SSVC --> SPOD2

    %% Storage bindings
    PGSTS --> PV1
    PGSTS --> PV2
    RDSTS --> RV1
    KFSTS --> KV1
    KFSTS --> KV2
    KFSTS --> KV3

    %% Configuration
    UPOD1 --> CM
    UPOD1 --> SEC
    QPOD1 --> CM
    QPOD1 --> SEC

    %% Ingress routing
    ING --> FSVC
    ING --> USVC
    ING --> QSVC
    ING --> ASVC
    ING --> SSVC

    classDef control fill:#ffebee,stroke:#c62828,stroke-width:2px
    classDef pod fill:#e1f5fe,stroke:#01579b,stroke-width:2px
    classDef storage fill:#f3e5f5,stroke:#4a148c,stroke-width:2px
    classDef config fill:#fff3e0,stroke:#e65100,stroke-width:2px

    class API,ETCD,SCHED,CTRL control
    class
FPOD1,FPOD2,UPOD1,UPOD2,QPOD1,QPOD2,QPOD3,APOD1,APOD2,SPOD1,SPOD2 pod
    class PGSTS,RDSTS,KFSTS,ZKSTS,PV1,PV2,RV1,KV1,KV2,KV3 storage
    class CM,SEC,ING config
```

## 2. Công cụ Triển khai

### 2.1 Container Orchestration Tools

#### A. Docker & Docker Compose

```
# Development environment
Tools:
  - Docker Engine: 24.0+
  - Docker Compose: 2.0+
  - Docker BuildKit: Enhanced build performance

Usage:
  - Local development
  - Integration testing
  - CI/CD pipeline builds
```

#### B. Docker Swarm

```
# Production ready clustering
Tools:
  - Docker Swarm Mode: Built-in orchestration
  - Docker Stack: Multi-service deployment
  - Docker Secrets: Configuration management

Benefits:
  - Simple setup compared to Kubernetes
  - Native Docker integration
  - Built-in load balancing
  - Rolling updates support
```

#### C. Kubernetes

```
# Enterprise grade orchestration
Tools:
  - Kubernetes: 1.28+
  - Helm: Package management
  - kubectl: CLI management
  - Kustomize: Configuration management

Distributions:
  - Google GKE: Managed Kubernetes
  - Amazon EKS: AWS managed
  - Azure AKS: Azure managed
  - Minikube: Local development
```

### 2.2 Infrastructure as Code Tools

#### A. Terraform

```
# Infrastructure provisioning
resource "aws_ecs_cluster" "quiz_app" {
  name = "quiz-app-cluster"

  setting {
    name  = "containerInsights"
    value = "enabled"
  }
}

resource "aws_ecs_service" "quiz_service" {
  name            = "quiz-service"
  cluster         = aws_ecs_cluster.quiz_app.id
  task_definition = aws_ecs_task_definition.quiz_service.arn
  desired_count   = 3

  load_balancer {
    target_group_arn = aws_lb_target_group.quiz_tg.arn
    container_name   = "quiz-service"
    container_port   = 3000
  }
}
```

#### B. Ansible

```
# Configuration management
- name: Deploy Quiz App Services
  hosts: all
  tasks:
    - name: Start Docker containers
      docker_container:
        name: "{{ item.name }}"
        image: "{{ item.image }}"
        ports: "{{ item.ports }}"
        env: "{{ item.environment }}"
      loop:
        - name: quiz-service
```

```
          image: quiz-app/quiz-service:latest
          ports: ["3000:3000"]
          environment:
            NODE_ENV: production
            DATABASE_URL: "{{ db_url }}"
```

### 2.3 CI/CD Pipeline Tools

#### A. GitLab CI/CD

```
# .gitlab-ci.yml
stages:
  - build
  - test
  - deploy

build_services:
  stage: build
  script:
    - docker build -t quiz-app/quiz-service:$CI_COMMIT_SHA services/quiz-
service/
    - docker push quiz-app/quiz-service:$CI_COMMIT_SHA

deploy_staging:
  stage: deploy
  script:
    - helm upgrade --install quiz-app ./helm-chart
      --set image.tag=$CI_COMMIT_SHA
      --namespace staging
  environment:
    name: staging
    url: https://staging.quiz-app.com
```

#### B. GitHub Actions

```
# .github/workflows/deploy.yml
name: Deploy Quiz App
on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Build and push images
        run: |
          docker build -t ${{ secrets.REGISTRY }}/quiz-service:${{
```

```
github.sha }} .
          docker push ${{ secrets.REGISTRY }}/quiz-service:${{ github.sha
}}

      - name: Deploy to Kubernetes
        run: |
          kubectl set image deployment/quiz-service
            quiz-service=${{ secrets.REGISTRY }}/quiz-service:${{
github.sha }}
```

## 3. Các bước Triển khai Hệ thống

### 3.1 Development Environment Setup

**Step 1: Local Development với Docker Compose**

```
# 1. Clone repository và setup environment
git clone https://github.com/your-org/quiz-app-event-driven.git
cd quiz-app-event-driven

# 2. Setup environment variables
cp .env.example .env
# Edit .env với local configuration

# 3. Build và start infrastructure services
docker-compose -f docker-compose.dev.yml up -d postgres redis kafka
zookeeper

# 4. Wait for services to be ready
./scripts/wait-for-services.sh

# 5. Run database migrations
npm run db:migrate

# 6. Build application images
docker-compose -f docker-compose.dev.yml build

# 7. Start all services
docker-compose -f docker-compose.dev.yml up -d

# 8. Verify deployment
curl http://localhost:3000/health
curl http://localhost:3001/health
curl http://localhost:3002/health
curl http://localhost:3003/health
```

**Step 2: Development Docker Compose Configuration**

```yaml
# docker-compose.dev.yml
version: '3.8'

services:
  # Infrastructure Services
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: quiz_dev
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./infra/postgres/init:/docker-entrypoint-initdb.d

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data

  zookeeper:
    image: confluentinc/cp-zookeeper:7.0.1
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  kafka:
    image: confluentinc/cp-kafka:7.0.1
    depends_on: [zookeeper]
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:9092
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1

  # Application Services
  user-service:
    build: ./services/user-service
    ports:
      - "3001:3001"
    environment:
      NODE_ENV: development
      DATABASE_URL: postgresql://postgres:password@postgres:5432/quiz_dev
    depends_on: [postgres]

  quiz-service:
    build: ./services/quiz-service
    ports:
      - "3000:3000"
    environment:
      NODE_ENV: development
      DATABASE_URL: postgresql://postgres:password@postgres:5432/quiz_dev
      REDIS_URL: redis://redis:6379
      KAFKA_BROKERS: kafka:9092
    depends_on: [postgres, redis, kafka]

  answer-service:
    build: ./services/answer-service
    ports:
      - "3002:3002"
    environment:
      NODE_ENV: development
      DATABASE_URL: postgresql://postgres:password@postgres:5432/quiz_dev
      KAFKA_BROKERS: kafka:9092
    depends_on: [postgres, kafka]

  scoring-service:
    build: ./services/scoring-service
    ports:
      - "3003:3003"
    environment:
      NODE_ENV: development
      DATABASE_URL: postgresql://postgres:password@postgres:5432/quiz_dev
      KAFKA_BROKERS: kafka:9092
    depends_on: [postgres, kafka]

  frontend:
    build: ./frontend
    ports:
      - "80:80"
    depends_on: [user-service, quiz-service, answer-service, scoring-
service]

volumes:
  postgres_data:
  redis_data:
```

### 3.2 Staging Environment Deployment

#### Step 1: Docker Swarm Staging Setup

```
# 1. Initialize Docker Swarm
docker swarm init --advertise-addr <MANAGER_IP>

# 2. Join worker nodes
docker swarm join --token <WORKER_TOKEN> <MANAGER_IP>:2377
```

```
# 3. Create overlay networks
docker network create --driver overlay quiz-app-network
docker network create --driver overlay kafka-network

# 4. Deploy stack services
docker stack deploy -c docker-stack.staging.yml quiz-app

# 5. Verify deployment
docker service ls
docker service ps quiz-app_quiz-service
```

#### Step 2: Staging Stack Configuration

```
# docker-stack.staging.yml
version: '3.8'

services:
  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    configs:
      - source: nginx_config
        target: /etc/nginx/nginx.conf
    deploy:
      replicas: 2
      update_config:
        parallelism: 1
        delay: 10s
    networks:
      - quiz-app-network

  quiz-service:
    image: quiz-app/quiz-service:staging
    environment:
      NODE_ENV: staging
      DATABASE_URL_FILE: /run/secrets/db_url
      KAFKA_BROKERS: kafka:9092
    secrets:
      - db_url
      - jwt_secret
    deploy:
      replicas: 3
      update_config:
        parallelism: 1
        delay: 30s
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
    networks:
```

```yaml
      - quiz-app-network
      - kafka-network

  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: quiz_staging
      POSTGRES_USER_FILE: /run/secrets/db_user
      POSTGRES_PASSWORD_FILE: /run/secrets/db_password
    secrets:
      - db_user
      - db_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    deploy:
      replicas: 1
      placement:
        constraints:
          - node.role == manager
    networks:
      - quiz-app-network

  kafka:
    image: confluentinc/cp-kafka:7.0.1
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:9092
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 3
    deploy:
      replicas: 3
      placement:
        max_replicas_per_node: 1
    networks:
      - kafka-network

networks:
  quiz-app-network:
    driver: overlay
    attachable: true
  kafka-network:
    driver: overlay

volumes:
  postgres_data:

secrets:
  db_url:
    external: true
  db_user:
    external: true
  db_password:
    external: true
  jwt_secret:
    external: true

configs:
  nginx_config:
    external: true
```

### 3.3 Production Kubernetes Deployment

#### Step 1: Kubernetes Production Setup

```
# 1. Setup Kubernetes cluster (example with kubeadm)
kubeadm init --pod-network-cidr=10.244.0.0/16

# 2. Install CNI plugin (Flannel)
kubectl apply -f https://raw.githubusercontent.com/flannel-
io/flannel/master/Documentation/kube-flannel.yml

# 3. Join worker nodes
kubeadm join <MASTER_IP>:6443 --token <TOKEN> --discovery-token-ca-cert-
hash <HASH>

# 4. Create namespace
kubectl create namespace quiz-app

# 5. Setup RBAC và service accounts
kubectl apply -f k8s/rbac/

# 6. Deploy infrastructure components
kubectl apply -f k8s/infrastructure/

# 7. Deploy application services
kubectl apply -f k8s/applications/

# 8. Setup ingress
kubectl apply -f k8s/ingress/
```

#### Step 2: Kubernetes Application Manifests

```
# k8s/applications/quiz-service.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: quiz-service
  namespace: quiz-app
  labels:
    app: quiz-service
spec:
  replicas: 3
```

```
    strategy:
      type: RollingUpdate
      rollingUpdate:
        maxSurge: 1
        maxUnavailable: 1
    selector:
      matchLabels:
        app: quiz-service
    template:
      metadata:
        labels:
          app: quiz-service
      spec:
        containers:
        - name: quiz-service
          image: quiz-app/quiz-service:v1.0.0
          ports:
          - containerPort: 3000
          env:
          - name: NODE_ENV
            value: "production"
          - name: DATABASE_URL
            valueFrom:
              secretKeyRef:
                name: quiz-secrets
                key: database-url
          - name: KAFKA_BROKERS
            value: "kafka:9092"
          - name: REDIS_URL
            value: "redis://redis:6379"
          resources:
            requests:
              memory: "256Mi"
              cpu: "250m"
            limits:
              memory: "512Mi"
              cpu: "500m"
          livenessProbe:
            httpGet:
              path: /health
              port: 3000
            initialDelaySeconds: 30
            periodSeconds: 10
          readinessProbe:
            httpGet:
              path: /ready
              port: 3000
            initialDelaySeconds: 5
            periodSeconds: 5
          volumeMounts:
          - name: config-volume
            mountPath: /app/config
        volumes:
        - name: config-volume
```

```
          configMap:
            name: quiz-config
---
apiVersion: v1
kind: Service
metadata:
  name: quiz-service
  namespace: quiz-app
spec:
  selector:
    app: quiz-service
  ports:
  - protocol: TCP
    port: 3000
    targetPort: 3000
  type: ClusterIP
```

**Step 3: Helm Chart Deployment**

```
# 1. Create Helm chart structure
helm create quiz-app

# 2. Customize values for production
cat > values.prod.yml << EOF
replicaCount: 3

image:
  repository: quiz-app/quiz-service
  tag: v1.0.0
  pullPolicy: IfNotPresent

resources:
  limits:
    cpu: 500m
    memory: 512Mi
  requests:
    cpu: 250m
    memory: 256Mi

autoscaling:
  enabled: true
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 80

ingress:
  enabled: true
  className: nginx
  hosts:
    - host: quiz-app.production.com
      paths:
        - path: /
```

```
          pathType: Prefix
  tls:
    - secretName: quiz-app-tls
      hosts:
        - quiz-app.production.com
EOF

# 3. Deploy with Helm
helm upgrade --install quiz-app ./quiz-app-chart \
  --values values.prod.yml \
  --namespace quiz-app \
  --create-namespace

# 4. Verify deployment
helm status quiz-app -n quiz-app
kubectl get pods -n quiz-app
```

### 3.4 Monitoring & Observability Setup

#### Step 1: Prometheus & Grafana Setup

```
# monitoring/prometheus-config.yml
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
    scrape_configs:
    - job_name: 'quiz-services'
      kubernetes_sd_configs:
      - role: pod
      relabel_configs:
      - source_labels: [__meta_kubernetes_pod_label_app]
        action: keep
        regex: quiz-service|answer-service|scoring-service|user-service
    - job_name: 'kafka'
      static_configs:
      - targets: ['kafka:9092']
    - job_name: 'postgres'
      static_configs:
      - targets: ['postgres:5432']
```

#### Step 2: Logging với ELK Stack

```
# logging/filebeat-config.yml
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: filebeat-config
data:
  filebeat.yml: |
    filebeat.inputs:
    - type: container
      paths:
        - /var/log/containers/*quiz*.log
      processors:
        - add_kubernetes_metadata:
            in_cluster: true

    output.elasticsearch:
      hosts: ["elasticsearch:9200"]
      index: "quiz-app-logs-%{+yyyy.MM.dd}"

    setup.template.name: "quiz-app"
    setup.template.pattern: "quiz-app-*"
```

**Step 3: Health Checks & Alerts**

```
# monitoring/alertmanager-rules.yml
groups:
- name: quiz-app-alerts
  rules:
  - alert: ServiceDown
    expr: up{job=~"quiz-.*"} == 0
    for: 1m
    labels:
      severity: critical
    annotations:
      summary: "Quiz service {{ $labels.instance }} is down"

  - alert: HighErrorRate
    expr: rate(http_requests_total{status=~"5.."}[5m]) > 0.1
    for: 2m
    labels:
      severity: warning
    annotations:
      summary: "High error rate detected in {{ $labels.service }}"

  - alert: KafkaLag
    expr: kafka_consumer_lag_sum > 1000
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: "Kafka consumer lag is high: {{ $value }}"
```

Kiến trúc triển khai Event-driven này giúp đảm bảo **high availability**, **scalability**, và **maintainability** cho Quiz App system, với khả năng tự động scale theo load và recover từ failures.

# Vẽ sơ đồ và giải thích góc nhìn giám sát (Observability) của hệ thống Event-driven trong bài lab04. Liệt kê các công cụ có thể sử dụng và các bước cần thực hiện để log, trace, và monitor các sự kiện từ lúc phát sinh đến lúc được xử lý.

## 1. Tổng quan Observability trong Event-Driven Architecture

### 1.1 Định nghĩa Observability

**Observability** là khả năng hiểu được trạng thái internal của hệ thống thông qua các outputs mà nó tạo ra. Trong Event-driven Architecture, observability trở nên cực kỳ quan trọng vì:

- **Distributed nature**: Events flow qua nhiều services và systems
- **Asynchronous processing**: Khó debug các issues realtime
- **Complex dependencies**: Service failures có thể cascade
- **Event ordering**: Cần track event sequence và causality

### 1.2 Three Pillars of Observability

**A. Metrics (Đo lường)**

**Mục đích**: Quantitative measurements của system performance **Đặc điểm**:

- Time-series data với timestamps
- Aggregated values (count, average, percentiles)
- Low storage overhead
- Good for alerting và monitoring trends

**B. Logs (Nhật ký)**

**Mục đích**: Detailed records của individual events **Đặc điểm**:

- Structured hoặc unstructured text
- Context-rich information
- High storage volume
- Good for debugging và forensics

**C. Traces (Dấu vết)**

**Mục đích**: End-to-end request flow tracking **Đặc điểm**:

- Distributed transaction visibility
- Span relationships và dependencies

- Performance bottleneck identification
- Root cause analysis

## 2. Sơ đồ Observability Architecture

### 2.1 Overall Observability View

```
graph LR
    subgraph "Application Layer"
        SERVICES[Quiz App Services
User, Quiz, Answer, Scoring]
        INFRA[Infrastructure
Kafka + PostgreSQL + Redis]
    end

    subgraph "Data Collection"
        METRICS[📊 Metrics
Prometheus Agents]
        LOGS[📄 Logs
Fluentd/Filebeat]
        TRACES[🔍 Traces
Jaeger/OpenTelemetry]
    end

    subgraph "Storage & Processing"
        PROMETHEUS[📈 Prometheus
Time-series DB]
        ELASTICSEARCH[🔎 Elasticsearch
Log Storage]
        JAEGER_DB[📊 Jaeger
Trace Storage]
    end

    subgraph "Visualization"
        GRAFANA[📱 Grafana
Dashboards & Alerts]
        KIBANA[🖥️ Kibana
Log Analysis]
    end

    %% Data flow
    SERVICES --> METRICS
    SERVICES --> LOGS
    SERVICES --> TRACES
    INFRA --> METRICS

    METRICS --> PROMETHEUS
    LOGS --> ELASTICSEARCH
    TRACES --> JAEGER_DB

    PROMETHEUS --> GRAFANA
    ELASTICSEARCH --> KIBANA
    JAEGER_DB --> GRAFANA

    %% Styling
    classDef app fill:#e3f2fd,stroke:#1976d2,stroke-width:2px
    classDef collect fill:#f3e5f5,stroke:#7b1fa2,stroke-width:2px
    classDef store fill:#e8f5e8,stroke:#388e3c,stroke-width:2px
    classDef viz fill:#fff3e0,stroke:#f57c00,stroke-width:2px

    class SERVICES,INFRA app
    class METRICS,LOGS,TRACES collect
    class PROMETHEUS,ELASTICSEARCH,JAEGER_DB store
    class GRAFANA,KIBANA viz
```

**Luồng đơn giản hóa của Observability:**

1. **Application Layer**: Quiz App services và infrastructure tạo ra data
2. **Data Collection**: Thu thập 3 loại data (Metrics, Logs, Traces)
3. **Storage & Processing**: Lưu trữ và xử lý data trong specialized databases
4. **Visualization**: Hiển thị insights thông qua dashboards và analytics tools

**Key Benefits của thiết kế này:**

- 📊 **Unified view**: Tất cả observability data trong một nơi
- 🔄 **Real-time**: Live monitoring của event flows
- 🎯 **Focused**: Tập trung vào 3 pillars chính
- 🚀 **Scalable**: Dễ dàng mở rộng từng component

### 2.2 Event Lifecycle Observability Flow

```
sequenceDiagram
    participant Client as Frontend Client
    participant QS as Quiz Service
    participant Kafka as Kafka Broker
    participant AS as Answer Service
    participant SS as Scoring Service
    participant Obs as Observability Stack

    Note over Client,Obs: Event: Quiz Started

    Client->>+QS: POST /quiz/start
    Note over QS: Generate Trace ID: trace-123
    QS->>Obs: Log: Quiz start requested
    QS->>Obs: Metric: quiz_start_requests_total++

    QS->>QS: Create QuizStartedEvent
    QS->>Obs: Log: Event created with ID: event-456
    QS->>Obs: Trace: Span started [quiz.start]

    QS->>+Kafka: Publish QuizStartedEvent
    QS->>Obs: Metric: kafka_events_published_total{topic=quiz-events}++
    Kafka->>Obs: Metric: kafka_topic_partition_current_offset++
    Kafka-->>QS: Ack
    QS->>Obs: Log: Event published successfully
    QS->>Obs: Trace: Span finished [quiz.start] duration=50ms

    QS-->>-Client: 200 OK {quizId}

    Note over Kafka,SS: Event Processing Phase

    Kafka->>+AS: Consume QuizStartedEvent
    AS->>Obs: Log: Event received, trace-id: trace-123
    AS->>Obs: Metric: kafka_events_consumed_total{topic=quiz-events}++
    AS->>Obs: Trace: Span started [answer.process] parent=trace-123

    AS->>AS: Process event (setup answer tracking)
    AS->>Obs: Log: Answer tracking initialized for quiz
    AS->>Obs: Metric: quiz_answer_tracking_initialized++
    AS->>Obs: Trace: Span finished [answer.process] duration=25ms
    AS-->>-Kafka: Commit offset

    Kafka->>+SS: Consume QuizStartedEvent
    SS->>Obs: Log: Event received, trace-id: trace-123
    SS->>Obs: Metric: kafka_events_consumed_total{topic=quiz-events}++
    SS->>Obs: Trace: Span started [scoring.process] parent=trace-123

    SS->>SS: Process event (setup leaderboard)
    SS->>Obs: Log: Leaderboard initialized for quiz
    SS->>Obs: Metric: leaderboard_initialized++
    SS->>Obs: Trace: Span finished [scoring.process] duration=30ms
    SS-->>-Kafka: Commit offset

    Note over Obs: Complete Event Trace Available
    Obs->>Obs: Correlate logs by trace-id: trace-123
    Obs->>Obs: Build distributed trace visualization
    Obs->>Obs: Calculate end-to-end latency: 105ms
```

## 3. Công cụ Observability cho Event-driven System

### 3.1 Metrics Collection & Storage

#### A. Prometheus Stack

**Mục đích**: Time-series metrics collection và alerting **Components**:

```
# Prometheus configuration
Prometheus Server:
  - Metrics scraping từ service endpoints
  - Rule evaluation cho alerting
  - Local storage (15 days retention)
  - PromQL query language

Node Exporter:
  - Host-level metrics (CPU, memory, disk, network)
  - System metrics cho infrastructure monitoring

Application Metrics:
  - Custom metrics từ Quiz App services
  - Business metrics (quiz created, answers submitted)
  - Performance metrics (request duration, error rates)
```

**Quiz App Metrics Examples**:

```
// services/quiz-service/src/metrics/quiz.metrics.ts
import { register, Counter, Histogram, Gauge } from 'prom-client';

export const quizMetrics = {
  quizzesCreated: new Counter({
    name: 'quiz_created_total',
    help: 'Total number of quizzes created',
    labelNames: ['status', 'host_id']
  }),

  quizDuration: new Histogram({
    name: 'quiz_duration_seconds',
    help: 'Quiz completion time in seconds',
    labelNames: ['quiz_id'],
    buckets: [30, 60, 120, 300, 600, 1200] // 30s to 20min
  }),

  activeQuizzes: new Gauge({
    name: 'active_quizzes_current',
    help: 'Number of currently active quizzes',
    labelNames: ['status']
  }),

  eventProcessingTime: new Histogram({
    name: 'event_processing_duration_ms',
    help: 'Time taken to process events',
    labelNames: ['event_type', 'service'],
    buckets: [1, 5, 10, 25, 50, 100, 250, 500, 1000]
  })
};
```

**B. Infrastructure Metrics**

**Kafka Metrics với JMX Exporter**:

```
# kafka-jmx-config.yml
rules:
- pattern: kafka.server<type=BrokerTopicMetrics, name=MessagesInPerSec,
topic=(.+)><>OneMinuteRate
```

```
  name: kafka_topic_messages_in_per_sec
  labels:
    topic: $1

- pattern: kafka.server<type=BrokerTopicMetrics, name=BytesInPerSec,
topic=(.+)><>OneMinuteRate
  name: kafka_topic_bytes_in_per_sec
  labels:
    topic: $1

- pattern: kafka.consumer<type=consumer-fetch-manager-metrics, client-id=
(.+), topic=(.+), partition=(.+)><>lag
  name: kafka_consumer_lag
  labels:
    client_id: $1
    topic: $2
    partition: $3
```

**PostgreSQL Metrics với postgres_exporter**:

```
# Custom queries cho Quiz App
- name: quiz_database_stats
  query: |
    SELECT
      schemaname,
      tablename,
      n_tup_ins as inserts,
      n_tup_upd as updates,
      n_tup_del as deletes,
      n_live_tup as live_tuples,
      n_dead_tup as dead_tuples
    FROM pg_stat_user_tables
    WHERE schemaname NOT IN ('information_schema', 'pg_catalog')
  metrics:
    - schemaname:
        usage: "LABEL"
    - tablename:
        usage: "LABEL"
    - inserts:
        usage: "COUNTER"
    - updates:
        usage: "COUNTER"
    - deletes:
        usage: "COUNTER"
    - live_tuples:
        usage: "GAUGE"
    - dead_tuples:
        usage: "GAUGE"
```

### 3.2 Logging Stack

### A. ELK/EFK Stack

**Elasticsearch**: Distributed search engine cho log storage **Logstash/Fluentd**: Log processing và enrichment **Kibana**: Log visualization và search interface

**Structured Logging cho Quiz App**:

```
// shared/utils/logger.ts
import winston from 'winston';
import { format } from 'winston';

const logFormat = format.combine(
  format.timestamp(),
  format.errors({ stack: true }),
  format.json(),
  format.printf(({ timestamp, level, message, service, traceId, spanId,
...meta }) => {
    return JSON.stringify({
      '@timestamp': timestamp,
      level,
      message,
      service,
      traceId,
      spanId,
      event: meta.event || null,
      userId: meta.userId || null,
      quizId: meta.quizId || null,
      duration: meta.duration || null,
      error: meta.error || null,
      ...meta
    });
  })
);

export const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || 'info',
  format: logFormat,
  transports: [
    new winston.transports.Console(),
    new winston.transports.File({
      filename: '/var/log/quiz-app/app.log',
      maxsize: 10485760, // 10MB
      maxFiles: 5
    })
  ]
});

// Event-specific logging
export const logEvent = (eventType: string, eventData: any, context: any =
{}) => {
  logger.info('Event processed', {
    event: {
      type: eventType,
      id: eventData.eventId,
      aggregateId: eventData.aggregateId,
      version: eventData.version,
      timestamp: eventData.timestamp
    },
    ...context
  });
};
```

**Fluentd Configuration cho Event Logs**:

```
# fluentd/fluent.conf
<source>
  @type tail
  path /var/log/quiz-app/*.log
  pos_file /var/log/fluentd/quiz-app.log.pos
  tag quiz-app.*
  format json
  time_key @timestamp
  time_format %Y-%m-%dT%H:%M:%S.%LZ
</source>

<filter quiz-app.**>
  @type parser
  key_name message
  reserve_data true
  <parse>
    @type json
  </parse>
</filter>

<filter quiz-app.**>
  @type record_transformer
  <record>
    hostname ${hostname}
    environment ${ENV}
    cluster_name quiz-app-cluster
  </record>
</filter>

# Enrich event logs với metadata
<filter quiz-app.** >
  @type grep
  <regexp>
    key event.type
    pattern ^(quiz\.|answer\.|score\.|user\.)
  </regexp>
</filter>

<match quiz-app.**>
  @type elasticsearch
  host elasticsearch
  port 9200
  index_name quiz-app-logs-%Y.%m.%d
  type_name _doc
  flush_interval 5s
  buffer_chunk_limit 2M
  buffer_queue_limit 32
</match>
```

**B. Log Parsing & Enrichment**

**Event Log Structure**:

```
{
  "@timestamp": "2024-01-15T10:30:45.123Z",
  "level": "info",
  "message": "Event processed successfully",
  "service": "quiz-service",
  "traceId": "trace-abc-123",
  "spanId": "span-def-456",
  "event": {
    "type": "quiz.started",
    "id": "event-789",
    "aggregateId": "quiz-xyz-999",
    "version": 1,
    "timestamp": "2024-01-15T10:30:45.000Z"
  },
  "userId": "user-123",
  "quizId": "quiz-xyz-999",
  "duration": 45,
  "hostname": "quiz-service-pod-1",
  "environment": "production",
  "cluster_name": "quiz-app-cluster"
}
```

## 3.3 Distributed Tracing

**A. OpenTelemetry Implementation**

**Instrumentation Setup**:

```
// shared/tracing/opentelemetry.ts
import { NodeSDK } from '@opentelemetry/auto-instrumentations-node';
import { Resource } from '@opentelemetry/resources';
import { SemanticResourceAttributes } from '@opentelemetry/semantic-
conventions';
import { JaegerExporter } from '@opentelemetry/exporter-jaeger';
import { BatchSpanProcessor } from '@opentelemetry/sdk-trace-base';
```

```
const jaegerExporter = new JaegerExporter({
  endpoint: process.env.JAEGER_ENDPOINT ||
'http://jaeger:14268/api/traces',
});

const sdk = new NodeSDK({
  resource: new Resource({
    [SemanticResourceAttributes.SERVICE_NAME]: process.env.SERVICE_NAME ||
'quiz-app',
    [SemanticResourceAttributes.SERVICE_VERSION]:
process.env.SERVICE_VERSION || '1.0.0',
    [SemanticResourceAttributes.DEPLOYMENT_ENVIRONMENT]:
process.env.NODE_ENV || 'development'
  }),
  spanProcessor: new BatchSpanProcessor(jaegerExporter),
  instrumentations: [] // Auto-instrumentation enabled
});

sdk.start();

// Custom tracing utilities
export const createEventSpan = (tracer: any, eventType: string,
parentContext?: any) => {
  return tracer.startSpan(`event.${eventType}`, {
    parent: parentContext,
    attributes: {
      'event.type': eventType,
      'component': 'event-processor'
    }
  });
};
```

**Event Processing Tracing**:

```
// services/quiz-service/src/services/quiz.service.ts
import { trace, context } from '@opentelemetry/api';
import { logger } from '../../../shared/utils/logger';

export class QuizService {
  private tracer = trace.getTracer('quiz-service');

  async createQuiz(createQuizDto: CreateQuizDto, hostId: string):
Promise<Quiz> {
    const span = this.tracer.startSpan('quiz.create');

    try {
      span.setAttributes({
        'quiz.title': createQuizDto.title,
        'quiz.host_id': hostId,
        'quiz.questions_count': createQuizDto.questions.length
      });
      // Business logic
      const quiz = await this.quizRepository.create({...createQuizDto,
hostId});

      // Create và publish event
      const event = new QuizCreatedEvent(quiz.id, quiz.title, hostId,
quiz.questions);

      // Propagate trace context to event
      const eventSpan = this.tracer.startSpan('event.quiz.created', {
parent: span });
      eventSpan.setAttributes({
        'event.id': event.eventId,
        'event.type': 'quiz.created',
        'event.aggregate_id': quiz.id
      });

      await context.with(trace.setSpan(context.active(), eventSpan), async
() => {
        await this.kafkaProducer.publishEvent(event);

        logger.info('Quiz created successfully', {
          quizId: quiz.id,
          eventId: event.eventId,
          traceId: span.spanContext().traceId,
          spanId: span.spanContext().spanId,
          duration: Date.now() - span.startTime
        });
      });

      eventSpan.end();
      span.setStatus({ code: 1 }); // OK

      return quiz;

    } catch (error) {
      span.recordException(error);
      span.setStatus({ code: 2, message: error.message }); // ERROR
      throw error;
    } finally {
      span.end();
    }
  }
}
```

**B. Cross-Service Trace Propagation**

**Kafka Message Headers cho Trace Context**:

```
// shared/services/kafka-producer.service.ts
import { trace, propagation, context } from '@opentelemetry/api';
```

```
export class KafkaProducerService {
  async publishEvent(event: DomainEvent): Promise<void> {
    const span = trace.getActiveSpan();
    const headers: Record<string, string> = {};

    // Inject trace context vào Kafka headers
    propagation.inject(context.active(), headers);

    const message = {
      key: event.aggregateId,
      value: JSON.stringify(event),
      headers: {
        ...headers,
        'event-type': event.eventType,
        'event-id': event.eventId,
        'aggregate-type': event.aggregateType,
        'content-type': 'application/json'
      }
    };

    await this.producer.send({
      topic: this.getTopicForEvent(event.eventType),
      messages: [message]
    });
  }
}

// Consumer side - extract trace context
export class KafkaConsumerService {
  @EventPattern('quiz.created')
  async handleQuizCreated(context: KafkaContext): Promise<void> {
    const message = context.getMessage();
    const headers = message.headers || {};

    // Extract trace context từ headers
    const parentContext = propagation.extract(context.active(), headers);

    await context.with(parentContext, async () => {
      const span = trace.getActiveSpan();
      const event = JSON.parse(message.value.toString());

      span?.setAttributes({
        'kafka.topic': context.getTopic(),
        'kafka.partition': context.getPartition(),
        'kafka.offset': context.getMessage().offset,
        'event.type': event.eventType,
        'event.id': event.eventId
      });

      await this.processQuizCreated(event);
    });
  }
}
```

## 4. Implementation Steps cho Event Observability

### 4.1 Step 1: Infrastructure Setup

**A. Docker Compose cho Observability Stack**

```
# docker-compose.observability.yml
version: '3.8'

services:
  # Metrics Stack
  prometheus:
    image: prom/prometheus:v2.40.0
    ports:
      - "9090:9090"
    volumes:
      -
./monitoring/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
      - ./monitoring/prometheus/rules:/etc/prometheus/rules
      - prometheus_data:/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--storage.tsdb.retention.time=15d'
      - '--web.console.libraries=/etc/prometheus/console_libraries'
      - '--web.console.templates=/etc/prometheus/consoles'
      - '--web.enable-lifecycle'
      - '--web.enable-admin-api'

  alertmanager:
    image: prom/alertmanager:v0.25.0
    ports:
      - "9093:9093"
    volumes:
      -
./monitoring/alertmanager/alertmanager.yml:/etc/alertmanager/alertmanager.
yml
      - alertmanager_data:/alertmanager

  # Logging Stack
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.5.0
    environment:
      - discovery.type=single-node
      - "ES_JAVA_OPTS=-Xms2g -Xmx2g"
      - xpack.security.enabled=false
    ports:
      - "9200:9200"
    volumes:
      - elasticsearch_data:/usr/share/elasticsearch/data
  kibana:
    image: docker.elastic.co/kibana/kibana:8.5.0
    ports:
      - "5601:5601"
    environment:
      - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
    depends_on:
      - elasticsearch

  logstash:
    image: docker.elastic.co/logstash/logstash:8.5.0
    ports:
      - "5044:5044"
      - "9600:9600"
    volumes:
      - ./monitoring/logstash/pipeline:/usr/share/logstash/pipeline
      -
./monitoring/logstash/logstash.yml:/usr/share/logstash/config/logstash.yml
    depends_on:
      - elasticsearch

  # Tracing Stack
  jaeger:
    image: jaegertracing/all-in-one:1.40
    ports:
      - "16686:16686" # Jaeger UI
      - "14268:14268" # HTTP collector
      - "14250:14250" # gRPC collector
    environment:
      - COLLECTOR_OTLP_ENABLED=true

  # Visualization
  grafana:
    image: grafana/grafana:9.3.0
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin123
    volumes:
      -
./monitoring/grafana/datasources:/etc/grafana/provisioning/datasources
      -
./monitoring/grafana/dashboards:/etc/grafana/provisioning/dashboards
      - grafana_data:/var/lib/grafana

volumes:
  prometheus_data:
  alertmanager_data:
  elasticsearch_data:
  grafana_data:
```

**B. Prometheus Configuration**

```yaml
# monitoring/prometheus/prometheus.yml
global:
  scrape_interval: 15s
  evaluation_interval: 15s

rule_files:
  - "rules/*.yml"

alerting:
  alertmanagers:
    - static_configs:
        - targets:
          - alertmanager:9093

scrape_configs:
  # Quiz App Services
  - job_name: 'quiz-services'
    static_configs:
      - targets:
        - 'user-service:3001'
        - 'quiz-service:3000'
        - 'answer-service:3002'
        - 'scoring-service:3003'
    metrics_path: /metrics
    scrape_interval: 10s

  # Infrastructure metrics
  - job_name: 'kafka'
    static_configs:
      - targets: ['kafka:9308'] # JMX exporter port
    scrape_interval: 30s

  - job_name: 'postgres'
    static_configs:
      - targets: ['postgres-exporter:9187']
    scrape_interval: 30s

  - job_name: 'redis'
    static_configs:
      - targets: ['redis-exporter:9121']
    scrape_interval: 30s

  # Node metrics
  - job_name: 'node'
    static_configs:
      - targets: ['node-exporter:9100']
    scrape_interval: 30s
```

### 4.2 Step 2: Application Instrumentation

### A. Metrics Implementation trong Services

```
// services/quiz-service/src/middleware/metrics.middleware.ts
import { Injectable, NestMiddleware } from '@nestjs/common';
import { Request, Response, NextFunction } from 'express';
import { register, Counter, Histogram } from 'prom-client';

@Injectable()
export class MetricsMiddleware implements NestMiddleware {
  private httpRequestsTotal = new Counter({
    name: 'http_requests_total',
    help: 'Total number of HTTP requests',
    labelNames: ['method', 'route', 'status_code'],
    registers: [register]
  });

  private httpRequestDuration = new Histogram({
    name: 'http_request_duration_ms',
    help: 'Duration of HTTP requests in milliseconds',
    labelNames: ['method', 'route', 'status_code'],
    buckets: [0.1, 5, 15, 50, 100, 500, 1000],
    registers: [register]
  });

  use(req: Request, res: Response, next: NextFunction) {
    const start = Date.now();

    res.on('finish', () => {
      const duration = Date.now() - start;
      const route = req.route?.path || req.path;

      this.httpRequestsTotal.inc({
        method: req.method,
        route,
        status_code: res.statusCode
      });

      this.httpRequestDuration.observe(
        { method: req.method, route, status_code: res.statusCode },
        duration
      );
    });

    next();
  }
}

// Metrics endpoint
@Controller('metrics')
export class MetricsController {
  @Get()
  async getMetrics(): Promise<string> {
    return await register.metrics();
  }
}
```

#### B. Event-specific Metrics

```
  // shared/metrics/event.metrics.ts
  import { Counter, Histogram, Gauge } from 'prom-client';

  export const eventMetrics = {
    // Event publishing metrics
    eventsPublished: new Counter({
      name: 'events_published_total',
      help: 'Total number of events published',
      labelNames: ['event_type', 'topic', 'service', 'status']
    }),

    eventPublishDuration: new Histogram({
      name: 'event_publish_duration_ms',
      help: 'Time taken to publish events',
      labelNames: ['event_type', 'topic', 'service'],
      buckets: [1, 5, 10, 25, 50, 100, 250, 500]
    }),

    // Event consumption metrics
    eventsConsumed: new Counter({
      name: 'events_consumed_total',
      help: 'Total number of events consumed',
      labelNames: ['event_type', 'topic', 'service', 'status']
    }),

    eventProcessingDuration: new Histogram({
      name: 'event_processing_duration_ms',
      help: 'Time taken to process events',
      labelNames: ['event_type', 'service', 'status'],
      buckets: [1, 5, 10, 25, 50, 100, 250, 500, 1000]
    }),

    // Event lag metrics
    eventLag: new Gauge({
      name: 'event_consumer_lag_seconds',
      help: 'Lag between event creation and processing',
      labelNames: ['event_type', 'consumer_group', 'topic', 'partition']
    }),

    // Business metrics
    quizEventsTotal: new Counter({
      name: 'quiz_events_total',
      help: 'Total quiz-related events',
      labelNames: ['event_type', 'quiz_id']
    }),
    activeQuizSessions: new Gauge({
      name: 'active_quiz_sessions',
      help: 'Number of active quiz sessions',
      labelNames: ['quiz_id']
    })
  };

  // Usage trong event handlers
  export const trackEventProcessing = (
    eventType: string,
    service: string,
    processingFn: () => Promise<void>
  ) => {
    return async () => {
      const startTime = Date.now();
      const timer = eventMetrics.eventProcessingDuration.startTimer({
        event_type: eventType,
        service: service
      });

      try {
        await processingFn();

        eventMetrics.eventsConsumed.inc({
          event_type: eventType,
          service: service,
          status: 'success'
        });

      } catch (error) {
        eventMetrics.eventsConsumed.inc({
          event_type: eventType,
          service: service,
          status: 'error'
        });
        throw error;
      } finally {
        timer();
      }
    };
  };
```

### 4.3 Step 3: Logging Implementation

**A. Structured Event Logging**

```
// shared/utils/event-logger.ts
import { logger } from './logger';
import { trace } from '@opentelemetry/api';
```

```
export interface EventLogContext {
  eventId: string;
  eventType: string;
  aggregateId: string;
  aggregateType: string;
  version: number;
  userId?: string;
  quizId?: string;
  correlationId?: string;
}

export class EventLogger {
  static logEventCreated(context: EventLogContext, payload: any) {
    const span = trace.getActiveSpan();
    const traceId = span?.spanContext().traceId;
    const spanId = span?.spanContext().spanId;

    logger.info('Event created', {
      ...context,
      traceId,
      spanId,
      payload: JSON.stringify(payload),
      phase: 'created',
      timestamp: new Date().toISOString()
    });
  }

  static logEventPublished(context: EventLogContext, topic: string,
partition?: number) {
    const span = trace.getActiveSpan();
    const traceId = span?.spanContext().traceId;
    const spanId = span?.spanContext().spanId;

    logger.info('Event published', {
      ...context,
      traceId,
      spanId,
      kafka: { topic, partition },
      phase: 'published',
      timestamp: new Date().toISOString()
    });
  }

  static logEventConsumed(context: EventLogContext, consumerGroup: string,
offset: string) {
    const span = trace.getActiveSpan();
    const traceId = span?.spanContext().traceId;
    const spanId = span?.spanContext().spanId;

    logger.info('Event consumed', {
      ...context,
      traceId,
      spanId,
      kafka: { consumerGroup, offset },
      phase: 'consumed',
      timestamp: new Date().toISOString()
    });
  }

  static logEventProcessed(context: EventLogContext, processingTime:
number, result: 'success' | 'error', error?: Error) {
    const span = trace.getActiveSpan();
    const traceId = span?.spanContext().traceId;
    const spanId = span?.spanContext().spanId;

    logger.info('Event processed', {
      ...context,
      traceId,
      spanId,
      processing: {
        duration: processingTime,
        result,
        error: error?.message
      },
      phase: 'processed',
      timestamp: new Date().toISOString()
    });
  }

  static logEventError(context: EventLogContext, error: Error, phase:
string) {
    const span = trace.getActiveSpan();
    const traceId = span?.spanContext().traceId;
    const spanId = span?.spanContext().spanId;

    logger.error('Event processing error', {
      ...context,
      traceId,
      spanId,
      error: {
        message: error.message,
        stack: error.stack,
        name: error.name
      },
      phase,
      timestamp: new Date().toISOString()
    });
  }
}
```

#### B. Quiz Service Event Logging Example

```
// services/quiz-service/src/services/quiz.service.ts
import { EventLogger } from '../../../shared/utils/event-logger';
import { eventMetrics } from '../../../shared/metrics/event.metrics';
```

```
export class QuizService {
  async createQuiz(createQuizDto: CreateQuizDto, hostId: string):
Promise<Quiz> {
    const quiz = await this.quizRepository.create({...createQuizDto,
hostId});

    // Create event
    const event = new QuizCreatedEvent(quiz.id, quiz.title, hostId,
quiz.questions);
    const eventContext: EventLogContext = {
      eventId: event.eventId,
      eventType: 'quiz.created',
      aggregateId: quiz.id,
      aggregateType: 'quiz',
      version: 1,
      userId: hostId,
      quizId: quiz.id
    };

    // Log event creation
    EventLogger.logEventCreated(eventContext, {
      title: quiz.title,
      questionsCount: quiz.questions.length
    });

    try {
      // Publish event
      const startTime = Date.now();
      await this.kafkaProducer.publishEvent(event);
      const publishTime = Date.now() - startTime;

      // Log successful publishing
      EventLogger.logEventPublished(eventContext, 'quiz-events');

      // Update metrics
      eventMetrics.eventsPublished.inc({
        event_type: 'quiz.created',
        topic: 'quiz-events',
        service: 'quiz-service',
        status: 'success'
      });

      eventMetrics.eventPublishDuration.observe({
        event_type: 'quiz.created',
        topic: 'quiz-events',
        service: 'quiz-service'
      }, publishTime);

    } catch (error) {
      EventLogger.logEventError(eventContext, error, 'publishing');

      eventMetrics.eventsPublished.inc({
        event_type: 'quiz.created',
        topic: 'quiz-events',
        service: 'quiz-service',
        status: 'error'
      });

      throw error;
    }

    return quiz;
  }
}
```

### 4.4 Step 4: End-to-End Event Tracing

#### A. Event Correlation ID System

```
// shared/types/event.types.ts
export interface TraceableEvent extends DomainEvent {
  traceId: string;
  spanId: string;
  parentSpanId?: string;
  correlationId: string;
  causationId?: string; // ID of the event that caused this event
  timestamp: string;
  metadata: {
    service: string;
    version: string;
    environment: string;
  };
}

// Event enrichment với trace context
export class TraceableEventFactory {
  static createEvent<T extends DomainEvent>(
    eventType: string,
    aggregateId: string,
    payload: any,
    causationEvent?: TraceableEvent
  ): TraceableEvent {
    const span = trace.getActiveSpan();
    const spanContext = span?.spanContext();

    return {
      eventId: uuidv4(),
      eventType,
      aggregateId,
      aggregateType: this.getAggregateType(aggregateId),
      payload,
      version: 1,
      traceId: spanContext?.traceId || uuidv4(),
      spanId: spanContext?.spanId || uuidv4(),
      parentSpanId: causationEvent?.spanId,
      correlationId: causationEvent?.correlationId || uuidv4(),
      causationId: causationEvent?.eventId,
      timestamp: new Date().toISOString(),
      metadata: {
        service: process.env.SERVICE_NAME || 'unknown',
        version: process.env.SERVICE_VERSION || '1.0.0',
        environment: process.env.NODE_ENV || 'development'
      }
    } as TraceableEvent;
  }
}
```

**B. Event Saga Tracing**

```
// services/scoring-service/src/sagas/quiz-completion.saga.ts
export class QuizCompletionSaga {
  @EventPattern('quiz.ended')
  async onQuizEnded(event: QuizEndedEvent): Promise<void> {
    const sagaId = `quiz-completion-${event.aggregateId}`;
    const state = new QuizCompletionState(sagaId, event.aggregateId);

    this.sagaStates.set(sagaId, state);

    // Step 1: Finalize all scores
    await this.finalizeScores(event.aggregateId);
    state.markScoresFinalized();

    // Step 2: Generate quiz report
    await this.generateQuizReport(event.aggregateId);
    state.markReportGenerated();

    // Step 3: Notify participants
    await this.notifyParticipants(event.aggregateId,
event.payload.participants);
    state.markParticipantsNotified();

    // Complete saga
    state.markCompleted();
    this.sagaStates.delete(sagaId);
  }

  @EventPattern('score.finalization.failed')
  async onScoreFinalizationFailed(event: ScoreFinalizationFailedEvent):
Promise<void> {
    const sagaId = `quiz-completion-${event.aggregateId}`;
    const state = this.sagaStates.get(sagaId);

    if (state) {
      // Compensating action: revert quiz to ended-with-errors state
      await this.revertQuizCompletion(event.aggregateId);
```

```
        state.markFailed('Score finalization failed');
      }
    }

    private async finalizeScores(quizId: string): Promise<void> {
      try {
        await this.kafkaProducer.publishEvent(
          new FinalizeScoresCommand(quizId)
        );
      } catch (error) {
        await this.kafkaProducer.publishEvent(
          new ScoreFinalizationFailedEvent(quizId, error.message)
        );
        throw error;
      }
    }

    private async generateQuizReport(quizId: string): Promise<void> {
      // Generate comprehensive quiz analytics
      const analytics = await
  this.analyticsService.generateQuizReport(quizId);

      await this.kafkaProducer.publishEvent(
        new QuizReportGeneratedEvent(quizId, analytics)
      );
    }

    private async notifyParticipants(quizId: string, participants:
  string[]): Promise<void> {
      const notifications = participants.map(playerId =>
        new ParticipantNotificationEvent(quizId, playerId, 'quiz_completed')
      );

      await this.kafkaProducer.publishBatch(notifications);
    }
  }
```

## 5. Monitoring Dashboards & Alerts

### 5.1 Grafana Dashboard Configuration

#### A. Event-driven System Overview Dashboard

```
{
  "dashboard": {
    "title": "Quiz App - Event-driven System Overview",
    "panels": [
      {
        "title": "Event Publishing Rate",
        "type": "graph",
        "targets": [
          {
            "expr": "rate(events_published_total[5m])",
            "legendFormat": "{{event_type}} - {{service}}"
          }
        ]
      },
      {
        "title": "Event Processing Latency",
        "type": "heatmap",
        "targets": [
          {
            "expr": "histogram_quantile(0.95,
rate(event_processing_duration_ms_bucket[5m]))",
            "legendFormat": "P95 - {{event_type}}"
          }
        ]
      },
      {
        "title": "Kafka Consumer Lag",
        "type": "graph",
        "targets": [
          {
            "expr": "kafka_consumer_lag",
            "legendFormat": "{{topic}} - {{partition}} -
{{consumer_group}}"
          }
        ]
      },
      {
        "title": "Event Error Rate",
        "type": "stat",
        "targets": [
          {
            "expr": "rate(events_consumed_total{status=\"error\"}[5m]) /
rate(events_consumed_total[5m]) * 100",
            "legendFormat": "Error Rate %"
          }
        ]
      }
    ]
  }
}
```

**B. Quiz Application Business Metrics Dashboard**

```
{
  "dashboard": {
    "title": "Quiz App - Business Metrics",
    "panels": [
      {
        "title": "Active Quizzes",
        "type": "stat",
        "targets": [
          {
            "expr": "active_quizzes_current",
            "legendFormat": "Active Quizzes"
          }
        ]
      },
      {
        "title": "Quiz Creation Rate",
        "type": "graph",
        "targets": [
          {
            "expr": "rate(quiz_created_total[5m])",
            "legendFormat": "Quizzes/min"
          }
        ]
      },
      {
        "title": "Answer Submission Rate",
        "type": "graph",
        "targets": [
          {
            "expr": "rate(answer_submitted_total[5m])",
            "legendFormat": "Answers/min"
          }
        ]
      },
      {
        "title": "Quiz Completion Time Distribution",
        "type": "heatmap",
        "targets": [
          {
            "expr": "histogram_quantile(0.50,
rate(quiz_duration_seconds_bucket[5m]))",
            "legendFormat": "P50"
          },
          {
            "expr": "histogram_quantile(0.95,
rate(quiz_duration_seconds_bucket[5m]))",
            "legendFormat": "P95"
          }
        ]
      }
    ]
  }
}
```

### 5.2 Alerting Rules

#### A. Event Processing Alerts

```
# monitoring/prometheus/rules/event-alerts.yml
groups:
- name: event-processing-alerts
  rules:
  - alert: HighEventProcessingLatency
    expr: histogram_quantile(0.95,
rate(event_processing_duration_ms_bucket[5m])) > 1000
    for: 2m
    labels:
      severity: warning
      team: platform
    annotations:
      summary: "High event processing latency detected"
      description: "Event processing P95 latency is {{ $value }}ms for {{
$labels.event_type }} in {{ $labels.service }}"

  - alert: EventProcessingErrorRate
    expr: rate(events_consumed_total{status="error"}[5m]) /
rate(events_consumed_total[5m]) > 0.05
    for: 1m
    labels:
      severity: critical
      team: platform
    annotations:
      summary: "High event processing error rate"
      description: "Event processing error rate is {{ $value |
humanizePercentage }}"

  - alert: KafkaLag
    expr: kafka_consumer_lag_sum > 1000
    for: 5m
    labels:
      severity: warning
      team: platform
    annotations:
      summary: "Kafka consumer lag is high: {{ $value }}"
```

**B. Business Logic Alerts**

```
# monitoring/prometheus/rules/business-alerts.yml
groups:
- name: business-alerts
  rules:
  - alert: QuizCreationRateDrop
    expr: rate(quiz_created_total[10m]) < 0.1
    for: 5m
    labels:
      severity: warning
      team: product
    annotations:
        summary: "Quiz creation rate dropped significantly"
        description: "Quiz creation rate is {{ $value }} quizzes/sec, below
normal threshold"

  - alert: HighAnswerSubmissionErrors
    expr: rate(answer_submitted_total{status="error"}[5m]) /
rate(answer_submitted_total[5m]) > 0.02
    for: 2m
    labels:
      severity: warning
      team: product
    annotations:
      summary: "High answer submission error rate"
      description: "Answer submission error rate is {{ $value |
humanizePercentage }}"

  - alert: QuizServiceDown
    expr: up{job="quiz-services", instance=~".*quiz-service.*"} == 0
    for: 30s
    labels:
      severity: critical
      team: platform
    annotations:
      summary: "Quiz service instance is down"
      description: "Quiz service instance {{ $labels.instance }} is not
responding"
```

## 6. Log Analysis & Troubleshooting

### 6.1 Kibana Queries cho Event Debugging

#### A. Event Flow Tracking

```
// Kibana search query cho tracking specific event flow
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "event.type": "quiz.created"
          }
        },
        {
          "range": {
            "@timestamp": {
              "gte": "now-1h",
              "lte": "now"
            }
          }
        }
      ]
    }
  },
  "sort": [
    {
      "@timestamp": {
        "order": "desc"
      }
    }
  ]
}

// Trace all events cho specific correlationId
{
  "query": {
    "match": {
      "correlationId": "correlation-abc-123"
    }
  },
  "sort": [
    {
      "@timestamp": {
        "order": "asc"
      }
    }
  ]
}
```

**B. Error Analysis Queries**

```
// Find failed event processing
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "level": "error"
          }
        },
        {
          "exists": {
            "field": "event.type"
          }
        }
      ]
    }
  },
  "aggs": {
    "error_by_event_type": {
      "terms": {
        "field": "event.type.keyword"
      }
    }
  }
}

// Performance issues tracking
{
  "query": {
    "bool": {
      "must": [
        {
          "range": {
            "duration": {
              "gte": 1000
            }
          }
        },
        {
          "match": {
            "phase": "processed"
          }
        }
      ]
    }
  }
}
```

## 6.2 Troubleshooting Playbooks

**A. Event Processing Delays**

```
# Troubleshooting steps cho event processing delays
Steps:
  1. Check Kafka consumer lag:
     Query: kafka_consumer_lag > threshold

  2. Verify service health:
     Query: up{job="quiz-services"} == 0

  3. Check event processing metrics:
     Query: rate(event_processing_duration_ms_bucket[5m])

  4. Analyze error logs:
     Kibana: level:error AND event.type:*

  5. Trace specific slow events:
     Jaeger: Search by operation "event.process" with high duration
```

**B. Event Ordering Issues**

```
# Troubleshooting event ordering problems
Steps:
  1. Check Kafka partition assignment:
     Log query: kafka.partition AND event.type:"quiz.*"

  2. Verify event versioning:
     Log query: event.version AND aggregateId:"quiz-123"

  3. Check causation chain:
     Log query: causationId:"event-456" OR eventId:"event-456"

  4. Trace event sequence:
     Jaeger: Search by correlationId to see full flow
```

## 7. Performance Monitoring & Optimization

### 7.1 Event Performance Metrics

#### A. Critical Performance KPIs

```
// Performance monitoring configuration
export const performanceKPIs = {
  // Event latency metrics
  eventE2ELatency: {
    name: 'event_end_to_end_latency_ms',
    description: 'Time from event creation to final processing',
    target: 'P95 < 500ms',
    alert_threshold: 1000
  },

  eventThroughput: {
    name: 'events_per_second',
    description: 'Number of events processed per second',
    target: '> 1000 events/sec',
    alert_threshold: 100
  },

  kafkaProducerLatency: {
    name: 'kafka_producer_latency_ms',
    description: 'Time to publish to Kafka',
    target: 'P95 < 50ms',
    alert_threshold: 100
  },

  consumerLag: {
    name: 'kafka_consumer_lag_messages',
    description: 'Messages waiting to be processed',
    target: '< 100 messages',
    alert_threshold: 1000
  }
};
```

#### B. Automated Performance Analysis

```
// services/monitoring-service/src/performance-analyzer.ts
export class PerformanceAnalyzer {
  async analyzeEventPerformance(timeRange: string = '1h'):
Promise<PerformanceReport> {
    const metrics = await this.prometheusClient.queryRange([
      `histogram_quantile(0.95,
rate(event_processing_duration_ms_bucket[5m]))`,
      `rate(events_published_total[5m])`,
      `rate(events_consumed_total[5m])`,
      `kafka_consumer_lag`
    ], timeRange);

    const report: PerformanceReport = {
      timestamp: new Date(),
      timeRange,
      eventLatency: {
        p50: metrics.eventLatency.p50,
        p95: metrics.eventLatency.p95,
        p99: metrics.eventLatency.p99
      },
      throughput: {
        published: metrics.publishRate,
        consumed: metrics.consumeRate,
        errorRate: metrics.errorRate
      },
      bottlenecks: await this.identifyBottlenecks(metrics),
      recommendations: await this.generateRecommendations(metrics)
    };

    // Auto-alert on performance degradation
    if (report.eventLatency.p95 >
performanceKPIs.eventE2ELatency.alert_threshold) {
      await this.alertManager.sendAlert('performance-degradation',
report);
    }

    return report;
  }

  private async identifyBottlenecks(metrics: any): Promise<Bottleneck[]> {
    const bottlenecks: Bottleneck[] = [];

    // Check for slow event processing
    if (metrics.eventLatency.p95 > 500) {
      bottlenecks.push({
        type: 'slow-processing',
        severity: 'high',
        description: `Event processing P95 latency is
${metrics.eventLatency.p95}ms`,
        affectedServices: await this.getSlowServices(metrics)
      });
    }

    // Check for consumer lag
    if (metrics.consumerLag.max > 1000) {
      bottlenecks.push({
        type: 'consumer-lag',
        severity: 'medium',
        description: `Consumer lag is ${metrics.consumerLag.max}
messages`,
        affectedTopics: await this.getLaggedTopics(metrics)
      });
    }

    return bottlenecks;
  }
}
```

Hệ thống Observability này cung cấp **end-to-end visibility** cho Event-driven Architecture của Quiz App, cho phép chúng ta:

- **Real-time monitoring** của event flows và performance
- **Root cause analysis** cho issues và failures
- **Proactive alerting** để prevent problems
- **Business insights** từ event data
- **Performance optimization** dựa trên data-driven decisions

Với implementation này, chúng ta có thể **maintain high reliability** và **quickly troubleshoot** các issues trong complex event-driven system.

# Liệt kê các đặc tính chất lượng mong muốn đạt được đối với Lambda Architecture. Liệt kê các công cụ có thể sử dụng và các bước cần thực hiện, để kiểm tra các đặc tính chất lương này. Vẽ và giải thích góc nhìn logic và góc nhìn process của Lambda Architecture ? Giải thích các công cụ sử dụng và từng bước để viết mã nguồn cho tính năng đếm lượt xem rất lớn của một video, với độ chính xác cao và khả năng truy vết nhằm mục đích thanh toán tiền, theo góc nhìn logic và process đề xuất.

## 1. Đặc tính Chất lượng mong muốn của Lambda Architecture

### A. Fault Tolerance (Khả năng chịu lỗi)

**Định nghĩa**: Hệ thống tiếp tục hoạt động đúng ngay cả khi có failures ở các components

**Cách đạt được trong Lambda Architecture**:

- **Batch Layer**: Immutable data store, có thể recompute từ raw data
- **Speed Layer**: Incremental processing với checkpointing
- **Serving Layer**: Distributed storage với replication
- **Data redundancy**: Raw data được lưu ở nhiều nơi

**Targets cho Video View Counting**:

- Data loss: 0% (không mất view counts)
- Recovery time: <5 minutes sau failures
- Availability: 99.99% uptime

### B. Scalability (Khả năng mở rộng)

**Định nghĩa**: Xử lý được lượng data và traffic tăng lên theo thời gian

**Horizontal scaling capabilities**:

- **Batch processing**: Có thể add thêm computing nodes
- **Stream processing**: Partition-based scaling
- **Storage**: Sharding và distributed storage
- **Serving**: Load balancing across multiple instances

**Scale targets**:

- Video views: 100M+ views/day
- Concurrent users: 1M+ simultaneous viewers
- Data throughput: 10GB+/hour raw events
- Query response time: <100ms at any scale

### C. Low Latency (Độ trễ thấp)

**Định nghĩa**: Cung cấp results gần real-time cho users

**Real-time requirements**:

- **Speed Layer**: Process events trong <1 second
- **Query serving**: Return view counts trong <100ms
- **Data freshness**: View counts updated trong 5 seconds
- **Payment accuracy**: Real-time validation cho revenue calculations

### D. Accuracy (Độ chính xác)

**Định nghĩa**: Đảm bảo correctness của data và calculations

**Accuracy mechanisms**:

- **Batch Layer**: Provides "ground truth" từ complete data
- **Speed Layer**: Approximate results với error bounds
- **Lambda merge**: Combine batch + speed results
- **Exactly-once processing**: Prevent duplicate counting

**Accuracy targets cho payment**:

- View counting accuracy: 99.99%
- Revenue calculation accuracy: 100% (critical for payments)
- Fraud detection: 95%+ accuracy
- Audit trail: 100% traceability

#### E. Extensibility (Khả năng mở rộng chức năng)

**Định nghĩa**: Dễ dàng thêm features mới và modify existing logic

**Benefits**:

- **Schema evolution**: Handle changing data formats
- **New algorithms**: Add advanced analytics without affecting existing
- **Multiple views**: Support different aggregation perspectives
- **A/B testing**: Easy to implement feature variations

## 2. Công cụ Kiểm tra Chất lượng

### A. Data Quality Testing

#### A.1 Batch Processing Quality Tools

**Apache Spark Testing Framework**:

```
# Data validation tests
spark-submit --class DataQualityTest \
  --master yarn \
  --deploy-mode cluster \
  batch-quality-test.jar
```

**Great Expectations cho Data Validation**:

```
# Video view data expectations
expectations = [
    "expect_column_values_to_not_be_null": ["video_id", "user_id",
"timestamp"],
    "expect_column_values_to_be_between": {
        "view_duration": {"min_value": 0, "max_value": 86400}
    },
    "expect_column_values_to_match_regex": {
        "video_id": "^[a-zA-Z0-9_-]+$"
    }
]
```

#### A.2 Stream Processing Quality Tools

**Kafka Streams Testing**:

```java
// Test topology for view counting
TopologyTestDriver testDriver = new TopologyTestDriver(topology, config);
TestInputTopic<String, ViewEvent> inputTopic =
    testDriver.createInputTopic("video-views", stringSerde,
viewEventSerde);
TestOutputTopic<String, ViewCount> outputTopic =
    testDriver.createOutputTopic("view-counts", stringSerde,
viewCountSerde);
```

### B. Performance Testing

#### B.1 Load Testing Tools

**Apache JMeter cho API Load Testing**:

```xml
<!-- JMeter test plan cho view API -->
<TestPlan>
  <ThreadGroup>
    <numThreads>1000</numThreads>
    <rampTime>60</rampTime>
    <duration>300</duration>
  </ThreadGroup>
  <HTTPSampler>
    <domain>api.video-platform.com</domain>
    <path>/videos/${videoId}/views</path>
    <method>GET</method>
  </HTTPSampler>
</TestPlan>
```

**Gatling cho High-Performance Testing**:

```scala
// Gatling scenario cho video view simulation
val viewScenario = scenario("Video View Load Test")
  .exec(http("Get View Count")
    .get("/api/videos/#{videoId}/views")
    .check(status.is(200))
    .check(responseTimeInMillis.lt(100)))
```

```
setUp(
  viewScenario.inject(rampUsers(10000) during (5 minutes))
).protocols(httpConf)
```

#### B.2 Stream Processing Performance

**Kafka Performance Testing**:

```
# Producer performance test
kafka-producer-perf-test.sh \
  --topic video-views \
  --num-records 1000000 \
  --record-size 512 \
  --throughput 50000 \
  --producer-props bootstrap.servers=localhost:9092

# Consumer performance test
kafka-consumer-perf-test.sh \
  --topic video-views \
  --messages 1000000 \
  --bootstrap-server localhost:9092
```

### C. Accuracy Testing

#### C.1 End-to-End Accuracy Validation

**Test Framework**:

```
# E2E accuracy test cho view counting
class ViewCountAccuracyTest:
    def test_batch_vs_speed_consistency(self):
        # Generate known test data
        test_events = self.generate_test_views(count=10000)

        # Process through speed layer
        speed_result = self.speed_layer.process(test_events)

        # Process through batch layer
        batch_result = self.batch_layer.process(test_events)

        # Compare results
        accuracy = self.compare_results(batch_result, speed_result)
        assert accuracy > 0.999  # 99.9% accuracy threshold

    def test_payment_calculation_accuracy(self):
        # Test revenue calculation accuracy
        views = self.load_test_views()
        calculated_revenue = self.revenue_calculator.calculate(views)
        expected_revenue = self.manual_calculation(views)
        assert abs(calculated_revenue - expected_revenue) < 0.01
```

## 3. Góc nhìn Logic của Lambda Architecture

### 3.1 Logical Architecture Overview

```
graph TB
    subgraph "Data Sources"
        WEB[Web Players]
        MOBILE[Mobile Apps]
        SMART[Smart TVs]
        API[API Clients]
    end

    subgraph "Data Ingestion"
        KAFKA[Kafka Event Streams]
    end

    subgraph "Lambda Architecture Core"
        subgraph "Batch Layer"
            HDFS[HDFS Raw Data Storage]
            SPARK[Spark Batch Processing]
            BATCH_DB[(Batch Views Database)]
        end

        subgraph "Speed Layer"
            STORM[Storm Real-time Processing]
            REDIS[(Redis Real-time Views)]
        end

        subgraph "Serving Layer"
            QUERY[Query Service]
            MERGE[Lambda Merge Logic]
        end
    end

    subgraph "Applications"
        ANALYTICS[Analytics Dashboard]
        BILLING[Billing System]
        CREATORS[Creator Portal]
    end

    %% Data flow
    WEB --> KAFKA
    MOBILE --> KAFKA
    SMART --> KAFKA
    API --> KAFKA

    KAFKA --> HDFS
    KAFKA --> STORM
    HDFS --> SPARK
    SPARK --> BATCH_DB

    STORM --> REDIS

    BATCH_DB --> QUERY
    REDIS --> QUERY
    QUERY --> MERGE

    MERGE --> ANALYTICS
    MERGE --> BILLING
    MERGE --> CREATORS

    classDef source fill:#e1f5fe,stroke:#01579b,stroke-width:2px
    classDef batch fill:#e8f5e8,stroke:#2e7d32,stroke-width:2px
    classDef speed fill:#fff3e0,stroke:#e65100,stroke-width:2px
    classDef serving fill:#f3e5f5,stroke:#4a148c,stroke-width:2px
    classDef app fill:#ffebee,stroke:#c62828,stroke-width:2px

    class WEB,MOBILE,SMART,API,KAFKA source
    class HDFS,SPARK,BATCH_DB batch
    class STORM,REDIS speed
    class QUERY,MERGE serving
    class ANALYTICS,BILLING,CREATORS app
```

### 3.2 Data Flow Logic

**A. Raw Data Schema**

```json
{
  "event_id": "uuid",
  "video_id": "string",
  "user_id": "string",
  "session_id": "string",
  "timestamp": "iso8601",
  "view_duration": "integer (seconds)",
  "device_type": "string",
  "geo_location": "string",
  "referrer": "string",
  "quality": "string (720p, 1080p, etc)",
  "revenue_eligible": "boolean",
  "ad_views": ["array of ad events"]
}
```

**B. Batch Layer Logic**

**Purpose**: Compute accurate, complete views từ all historical data **Processing**: Daily/hourly batch jobs tính toán view counts **Output**: Authoritative view counts cho billing và analytics

**C. Speed Layer Logic**

**Purpose**: Provide real-time updates cho current view counts **Processing**: Stream processing của live events **Output**: Incremental view counts với low latency

**D. Serving Layer Logic**

**Purpose**: Merge batch và speed layer results **Query Logic**:

```
Total Views = Batch Views (up to last batch) + Speed Views (since last
batch)
```

## 4. Góc nhìn Process của Lambda Architecture

### 4.1 Process Flow Diagram

```
sequenceDiagram
    participant User as Video Viewer
    participant App as Video App
    participant Kafka as Kafka Stream
    participant Speed as Speed Layer
    participant Batch as Batch Layer
    participant Serve as Serving Layer
    participant Bill as Billing System

    Note over User,Bill: Video View Process Flow

    User->>App: Watch Video
    App->>Kafka: Publish View Event

    par Real-time Processing
        Kafka->>Speed: Stream Event
        Speed->>Speed: Update Real-time Count
        Speed->>Serve: Push Live Count
    and Batch Processing
        Kafka->>Batch: Store Raw Event
        Note over Batch: Every Hour/Day
        Batch->>Batch: Recompute All Views
        Batch->>Serve: Update Authoritative Count
    end

    App->>Serve: Query Current Views
    Serve->>Serve: Merge Batch + Speed
    Serve-->>App: Return Total Views

    Note over Bill: Payment Process
    Bill->>Serve: Query Revenue-eligible Views
    Serve->>Serve: Apply Payment Logic
    Serve-->>Bill: Return Billable Views
    Bill->>Bill: Calculate Payment
```

### 4.2 Detailed Process Workflows

#### A. View Event Processing Workflow

```
flowchart TD
    START([Video View Starts]) --> VALIDATE{Valid Event?}
    VALIDATE -->|Yes| KAFKA[Send to Kafka]
    VALIDATE -->|No| DROP[Drop Event]

    KAFKA --> SPEED[Speed Layer Processing]
    KAFKA --> BATCH[Batch Layer Storage]

    SPEED --> REALTIME[Update Real-time Count]
    REALTIME --> CACHE[Update Cache]

    BATCH --> SCHEDULE{Batch Schedule?}
    SCHEDULE -->|Yes| COMPUTE[Recompute Views]
    SCHEDULE -->|No| WAIT[Wait for Schedule]

    COMPUTE --> VALIDATE_BATCH{Validation Pass?}
    VALIDATE_BATCH -->|Yes| UPDATE[Update Batch Views]
    VALIDATE_BATCH -->|No| RETRY[Retry Computation]

    UPDATE --> SERVE[Update Serving Layer]
    CACHE --> SERVE

    SERVE --> API[Expose via API]
    API --> END([Process Complete])

    classDef process fill:#e3f2fd,stroke:#1976d2,stroke-width:2px
    classDef decision fill:#fff3e0,stroke:#f57c00,stroke-width:2px
    classDef storage fill:#e8f5e8,stroke:#388e3c,stroke-width:2px

    class START,END process
    class VALIDATE,SCHEDULE,VALIDATE_BATCH decision
    class KAFKA,CACHE,SERVE storage
```

## 5. Công cụ Implementation cho Video View Counting

### 5.1 Technology Stack

#### A. Data Ingestion Layer

```
Apache Kafka:
  Purpose: Real-time event streaming
  Configuration:
    - Topics: video-views, ad-views, user-sessions
    - Partitions: 12 (để scale processing)
    - Replication: 3 (cho fault tolerance)
    - Retention: 7 days (cho batch reprocessing)

Kafka Connect:
  Purpose: Data integration với external systems
  Connectors:
    - HDFS Sink: Store raw events
    - Elasticsearch Sink: Search và analytics
```

**B. Batch Layer Stack**

```
Apache Hadoop HDFS:
  Purpose: Distributed storage cho raw data
  Configuration:
    - Block size: 128MB
    - Replication factor: 3
    - Compression: Snappy

Apache Spark:
  Purpose: Large-scale batch processing
  Configuration:
    - Cluster mode: YARN
    - Executors: 20 nodes
    - Memory per executor: 8GB
    - Cores per executor: 4
```

**C. Speed Layer Stack**

```
Apache Storm:
  Purpose: Real-time stream processing
  Configuration:
    - Workers: 10 nodes
    - Parallelism: 48 threads
    - Message timeout: 30 seconds

Redis Cluster:
  Purpose: Fast access cho real-time data
  Configuration:
    - Nodes: 6 (3 masters, 3 slaves)
    - Memory: 16GB per node
    - Persistence: RDB + AOF
```

### 5.2 Simple Code Implementation

### A. View Event Data Model

```python
# models/view_event.py
from dataclasses import dataclass
from datetime import datetime
from typing import Optional

@dataclass
class ViewEvent:
    event_id: str
    video_id: str
    user_id: str
    timestamp: datetime
    view_duration: int  # seconds
    device_type: str
    revenue_eligible: bool

    def to_dict(self):
        return {
            'event_id': self.event_id,
            'video_id': self.video_id,
            'user_id': self.user_id,
            'timestamp': self.timestamp.isoformat(),
            'view_duration': self.view_duration,
            'device_type': self.device_type,
            'revenue_eligible': self.revenue_eligible
        }
```

### B. Kafka Producer (Simple)

```python
# ingestion/kafka_producer.py
from kafka import KafkaProducer
import json

class ViewEventProducer:
    def __init__(self):
        self.producer = KafkaProducer(
            bootstrap_servers=['localhost:9092'],
            value_serializer=lambda v: json.dumps(v).encode('utf-8')
        )

    def send_view_event(self, view_event: ViewEvent):
        try:
            # Send to video-views topic
            self.producer.send(
                'video-views',
                key=view_event.video_id.encode('utf-8'),
                value=view_event.to_dict()
            )
            print(f"Sent view event: {view_event.event_id}")
        except Exception as e:
            print(f"Error sending event: {e}")
```

### C. Speed Layer Processing (Simple)

```python
# speed_layer/storm_processor.py
import redis
from kafka import KafkaConsumer
import json

class RealTimeViewProcessor:
    def __init__(self):
        self.redis_client = redis.Redis(host='localhost', port=6379, db=0)
        self.consumer = KafkaConsumer(
            'video-views',
            bootstrap_servers=['localhost:9092'],
            value_deserializer=lambda m: json.loads(m.decode('utf-8'))
        )

    def process_events(self):
        for message in self.consumer:
            view_event = message.value
            video_id = view_event['video_id']

            # Update real-time view count
            current_count = self.redis_client.get(f"views:{video_id}")
            if current_count:
                new_count = int(current_count) + 1
            else:
                new_count = 1

            self.redis_client.set(f"views:{video_id}", new_count)

            # Update revenue-eligible views
            if view_event['revenue_eligible']:
                revenue_key = f"revenue_views:{video_id}"
                self.redis_client.incr(revenue_key)

            print(f"Updated views for {video_id}: {new_count}")
```

### D. Batch Layer Processing (Simple)

```python
# batch_layer/spark_processor.py
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

class BatchViewProcessor:
    def __init__(self):
        self.spark = SparkSession.builder \
            .appName("VideoViewBatchProcessor") \
            .getOrCreate()

    def process_daily_views(self, date):
        # Read raw events from HDFS
        events_df = self.spark.read.json(f"hdfs://raw-
events/{date}/*.json")

        # Calculate total views per video
        view_counts = events_df.groupBy("video_id") \
            .agg(
                count("*").alias("total_views"),
                sum(when(col("revenue_eligible") == True,
1).otherwise(0)).alias("revenue_views"),
                avg("view_duration").alias("avg_duration")
            )

        # Save to batch views table
        view_counts.write \
            .mode("overwrite") \
            .option("path", f"hdfs://batch-views/{date}") \
            .saveAsTable("video_views_batch")

        print(f"Processed batch views for {date}")
```

**E. Serving Layer (Simple)**

```
# serving_layer/query_service.py
import redis
from pyspark.sql import SparkSession

class ViewCountService:
    def __init__(self):
        self.redis_client = redis.Redis(host='localhost', port=6379, db=0)
        self.spark =
SparkSession.builder.appName("QueryService").getOrCreate()

    def get_total_views(self, video_id):
        # Get batch views (up to yesterday)
        batch_df = self.spark.sql(f"""
            SELECT total_views, revenue_views
            FROM video_views_batch
            WHERE video_id = '{video_id}'
        """)

        batch_views = 0
        batch_revenue = 0
        if batch_df.count() > 0:
            row = batch_df.first()
```

```
                batch_views = row['total_views']
                batch_revenue = row['revenue_views']

        # Get real-time views (today)
        speed_views = self.redis_client.get(f"views:{video_id}")
        speed_revenue = self.redis_client.get(f"revenue_views:{video_id}")

        speed_views = int(speed_views) if speed_views else 0
        speed_revenue = int(speed_revenue) if speed_revenue else 0

        return {
            'video_id': video_id,
            'total_views': batch_views + speed_views,
            'revenue_views': batch_revenue + speed_revenue,
            'batch_views': batch_views,
            'realtime_views': speed_views
        }
```

#### F. Payment Calculation Service (Simple)

```
# billing/payment_service.py
class PaymentCalculator:
    def __init__(self):
        self.view_service = ViewCountService()

    def calculate_revenue(self, video_id, rate_per_view=0.001):
        view_data = self.view_service.get_total_views(video_id)
        revenue_views = view_data['revenue_views']

        # Calculate payment
        total_revenue = revenue_views * rate_per_view

        # Audit trail
        audit_record = {
            'video_id': video_id,
            'revenue_views': revenue_views,
            'rate_per_view': rate_per_view,
            'total_revenue': total_revenue,
            'calculation_timestamp': datetime.now().isoformat(),
            'batch_views': view_data['batch_views'],
            'realtime_views': view_data['realtime_views']
        }

        # Store audit trail
        self.store_audit_record(audit_record)

        return total_revenue

    def store_audit_record(self, record):
        # Store in audit database for traceability
        pass  # Implementation depends on audit storage choice
```

## 6. Deployment và Monitoring

### 6.1 Simple Deployment

```yaml
# docker-compose.yml
version: '3.8'
services:
  kafka:
    image: confluentinc/cp-kafka:latest
    ports:
      - "9092:9092"
    environment:
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092

  redis:
    image: redis:alpine
    ports:
      - "6379:6379"

  spark:
    image: bitnami/spark:latest
    ports:
      - "8080:8080"
    environment:
      SPARK_MODE: master
```

### 6.2 Monitoring Setup

```python
# monitoring/metrics.py
from prometheus_client import Counter, Histogram, Gauge

# Metrics for monitoring
view_events_total = Counter('view_events_total', 'Total view events processed')
processing_latency = Histogram('processing_latency_seconds', 'Event processing latency')
active_viewers = Gauge('active_viewers', 'Current active viewers')

def track_view_event():
    view_events_total.inc()

def track_processing_time(duration):
    processing_latency.observe(duration)
```