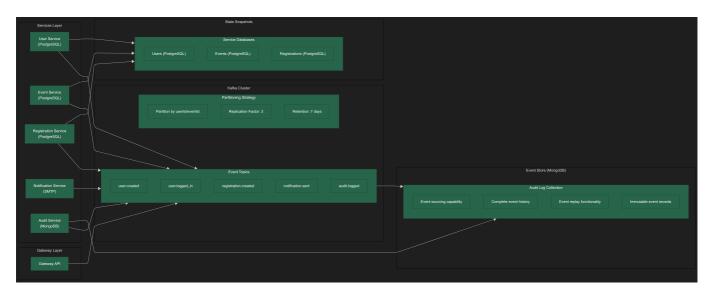
# 🗐 Event-Driven Architecture Analysis - Lab 04

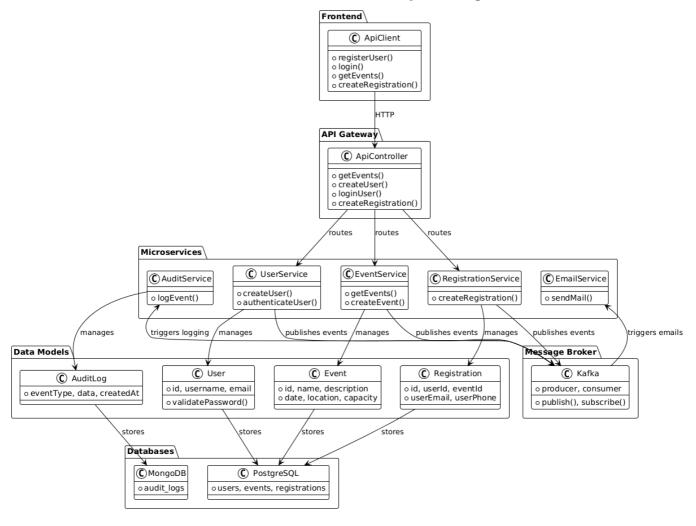
# 🌠 Sơ Đồ Kiến Trúc (Architecture Diagrams)

1. Sơ đồ lưu trữ sự kiện (Event Store Architecture)

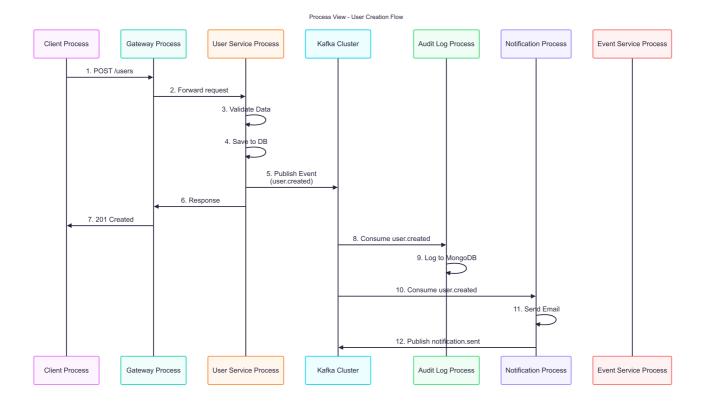


2. Sơ đồ logic tổng thể (Logical Architecture)

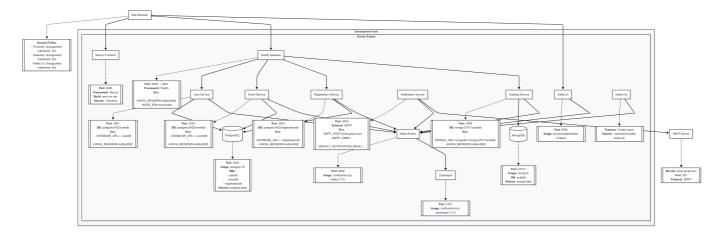
### **Event-Driven Architecture Demo - Simplified Logical View**



## 3. Sơ đồ luồng xử lý (Process View)

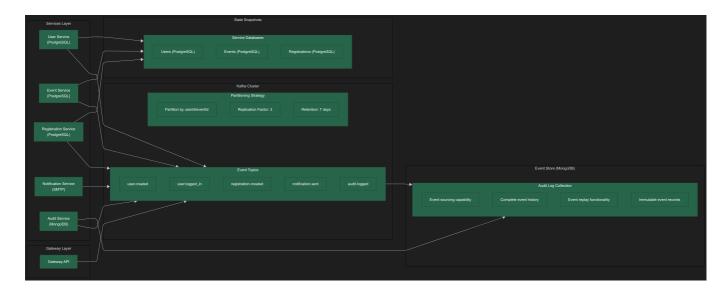


# 4. Sơ đồ triển khai (Deployment Architecture)



# © Câu 14: Đặc Tính Chất Lượng và Công Cụ Kiểm Tra

■ Sơ Đồ Lưu Trữ Event-Driven Architecture



🔧 Công Cụ và Bước Triển Khai Sơ Đồ Lưu Trữ Sự Kiện

# Danh Sách Công Cụ Cần Thiết

🋠 Công Cụ	Mục Đích	📥 Cài Đặt
Apache Kafka	Message Broker chính	docker pull confluentinc/cp-kafka:7.3.0
Apache Zookeeper	Quản lý metadata Kafka	docker pull confluentinc/cp-zookeeper:7.3.0
MongoDB	Event Store Database	docker pull mongo:6.0
PostgreSQL	Service Databases	docker pull postgres:15
Redis	Caching & Session	docker pull redis:7-alpine
Kafka UI	Kafka Management	docker pull provectuslabs/kafka-ui:latest

# T Bước 1: Cài Đặt Cơ Sở Hạ Tầng

## 1.1 Docker Compose Configuration

```
- zookeeper_logs:/var/lib/zookeeper/log
  # Kafka Broker
  kafka:
    image: confluentinc/cp-kafka:7.3.0
    hostname: kafka
    container_name: kafka
    depends on:
      zookeeper
    ports:
      - "29092:29092"
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT: PLAINTEXT, PLAINTEXT_HOST: PLAINTEXT
      KAFKA ADVERTISED LISTENERS:
PLAINTEXT://kafka:29092,PLAINTEXT HOST://localhost:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      # Event Storage Configuration
      KAFKA_LOG_RETENTION_HOURS: 168
                                             # 7 ngày
      KAFKA_LOG_RETENTION_BYTES: 1073741824 # 1GB
      KAFKA_LOG_SEGMENT_BYTES: 1073741824 # 1GB
      KAFKA NUM PARTITIONS: 3
      KAFKA_DEFAULT_REPLICATION_FACTOR: 1
    volumes:
      - kafka data:/var/lib/kafka/data
    healthcheck:
      test: kafka-broker-api-versions --bootstrap-server kafka:29092
      interval: 10s
      timeout: 5s
      retries: 5
  # MongoDB cho Event Store
  mongodb:
    image: mongo:6.0
    container_name: mongodb
    restart: always
    ports:
      - "27017:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin
      MONGO_INITDB_ROOT_PASSWORD: admin123
      MONGO_INITDB_DATABASE: eventstore
    volumes:
      - mongodb_data:/data/db
      - mongodb_config:/data/configdb
      - ./mongo-init.js:/docker-entrypoint-initdb.d/mongo-init.js:ro
  # PostgreSQL cho Service Databases
```

```
postgres:
    image: postgres:15
    container_name: postgres
    restart: always
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: postgres
      POSTGRES PASSWORD: admin123
      POSTGRES_DB: eda_demo
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./init-postgres.sql:/docker-entrypoint-initdb.d/init-postgres.sql
  # Redis cho Caching
  redis:
    image: redis:7-alpine
    container name: redis
    restart: always
    ports:
      - "6379:6379"
    command: redis-server --appendonly yes --requirepass redis123
    volumes:
      - redis_data:/data
  # Kafka UI cho quản lý
  kafka-ui:
    image: provectuslabs/kafka-ui:latest
    container_name: kafka-ui
    depends_on:
      - kafka
    ports:
      - "8080:8080"
    environment:
      KAFKA_CLUSTERS_0_NAME: local
      KAFKA_CLUSTERS_0_B00TSTRAPSERVERS: kafka:29092
      KAFKA_CLUSTERS_0_Z00KEEPER: zookeeper:2181
volumes:
  zookeeper_data:
  zookeeper_logs:
  kafka_data:
  mongodb_data:
  mongodb_config:
  postgres_data:
  redis_data:
```

### 1.2 Database Initialization Scripts

```
-- init-postgres.sql: Khởi tạo databases cho các services
CREATE DATABASE userdb;
```

```
CREATE DATABASE eventdb;
CREATE DATABASE registrationdb;
CREATE DATABASE auditdb;

-- Tạo users cho từng service
CREATE USER user_service WITH ENCRYPTED PASSWORD 'user123';
CREATE USER event_service WITH ENCRYPTED PASSWORD 'event123';
CREATE USER registration_service WITH ENCRYPTED PASSWORD 'reg123';

-- Phân quyền
GRANT ALL PRIVILEGES ON DATABASE userdb TO user_service;
GRANT ALL PRIVILEGES ON DATABASE eventdb TO event_service;
GRANT ALL PRIVILEGES ON DATABASE registrationdb TO registration_service;
```

```
// mongo-init.js: Khởi tao MongoDB cho Event Store
db = db.getSiblingDB('eventstore');
// Tao collections
db.createCollection('events');
db.createCollection('snapshots');
// Tao indexes cho performance
db.events.createIndex({ "aggregateId": 1, "version": 1 });
db.events.createIndex({ "eventType": 1, "timestamp": 1 });
db.events.createIndex({ "timestamp": 1 });
// Tao user cho services
db.createUser({
 user: "event_writer",
  pwd: "eventwrite123",
  roles: [
    { role: "readWrite", db: "eventstore" }
  ]
});
print("MongoDB Event Store initialized successfully");
```

# 🚀 Bước 2: Khởi Tạo Kafka Topics

```
#!/bin/bash
# scripts/init-kafka-topics.sh

echo "☑ Initializing Kafka Topics..."

# Đợi Kafka sẵn sàng
echo "☒ Waiting for Kafka to be ready..."
until docker exec kafka kafka-broker-api-versions --bootstrap-server
localhost:29092 > /dev/null 2>&1; do
echo "Kafka is not ready yet..."
```

```
sleep 2
done
echo "√ Kafka is ready! Creating topics..."
# Tao topics cho từng domain
docker exec kafka kafka-topics --create \
  --topic user.created \
  --bootstrap-server localhost:29092 \
  --partitions 3 \
  --replication-factor 1 \
  --config retention.ms=604800000 \
  --config segment.ms=86400000
docker exec kafka kafka-topics --create \
  --topic user.updated \
  --bootstrap-server localhost:29092 \
  --partitions 3 \
  --replication-factor 1 \
  --config retention.ms=604800000
docker exec kafka kafka-topics --create \
  --topic user.logged_in \
  --bootstrap-server localhost:29092 \
 --partitions 3 \
  --replication-factor 1 \
  --config retention.ms=604800000
docker exec kafka kafka-topics --create \
  --topic registration.created \
  --bootstrap-server localhost:29092 \
  --partitions 3 \
  --replication-factor 1
docker exec kafka kafka-topics --create \
  --topic notification.sent \
  --bootstrap-server localhost:29092 \
  --partitions 3 \
  --replication-factor 1
docker exec kafka kafka-topics --create \
  --topic audit.logged \
  --bootstrap-server localhost:29092 \
  --partitions 1 \
  --replication-factor 1 \
  --config cleanup.policy=compact
echo " Listing all topics:"
docker exec kafka kafka-topics --list --bootstrap-server localhost:29092
echo " Kafka topics initialized successfully!"
```

# H Bước 3: Triển Khai Event Publisher/Consumer (Đơn giản)

## 3.1 Event Publisher - Xuất Bản Sự Kiện

```
// services/eventPublisher.js - Gửi events qua Kafka
const kafka = require('kafkajs');
class EventPublisher {
  constructor() {
    this kafka = kafka({
      clientId: 'event-publisher',
      brokers: ['localhost:9092']
    }):
   this.producer = this.kafka.producer();
  }
  async publishEvent(topic, eventData) {
    const message = {
      key: eventData.id || Date.now().toString(),
      value: JSON.stringify({
        ...eventData,
        timestamp: new Date().toISOString()
      })
    };
    await this.producer.send({
     topic,
     messages: [message]
    });
    console.log(`Event published to ${topic}:`, eventData);
  async disconnect() {
    await this.producer.disconnect();
}
module.exports = EventPublisher;
```

## 3.2 Event Consumer - Tiêu Thụ Sự Kiện

```
// services/eventConsumer.js - Nhận events từ Kafka
class EventConsumer {
  constructor(groupId) {
    this.kafka = kafka({
      clientId: 'event-consumer',
       brokers: ['localhost:9092']
    });
  this.consumer = this.kafka.consumer({ groupId });
```

```
async subscribe(topics, handler) {
   await this.consumer.subscribe({ topics });

await this.consumer.run({
   eachMessage: async ({ topic, message }) => {
      const event = JSON.parse(message.value.toString());
      await handler(topic, event);
   }
   });
}

async disconnect() {
   await this.consumer.disconnect();
}

module.exports = EventConsumer;
```

```
type: Number,
required: true,
min: 1
```

},

// Thời điểm sự kiện xảy ra timestamp: { type: Date, default: Date.now, index: true },

// Metadata bổ sung metadata: { userld: String, // Ai thực hiện hành động requestld: String, // Request ID để tracking source: String, // Service nào tạo event correlationld: String // Để trace qua services } }, { // Tự động tạo \_id và versionKey timestamps: false, versionKey: false });

// Compound index cho performance eventStoreSchema.index({ aggregateId: 1, eventVersion: 1 }); eventStoreSchema.index({ eventType: 1, timestamp: -1 });

// Đảm bảo unique constraint cho (aggregateld + eventVersion) eventStoreSchema.index( { aggregateld: 1, eventVersion: 1}, { unique: true } );

module.exports = mongoose.model('Event', eventStoreSchema);

```
### **3.2 Event Writer Service**
```javascript
// services/eventWriter.js - Ghi sự kiện vào Event Store
const Event = require('../models/eventStore');
const kafkaProducer = require('./kafkaProducer');

class EventWriter {
  constructor() {
```

```
this.producer = kafkaProducer;
 }
 /**
  * Ghi môt sư kiến vào Event Store và publish lên Kafka
  * @param {string} aggregateId - ID của aggregate
  * @param {string} eventType - Loại sự kiện
  * @param {object} eventData - Dữ liêu sư kiên
  * @param {object} metadata - Metadata
 async writeEvent(aggregateId, eventType, eventData, metadata = {}) {
   try {
     // 1. Lấy version tiếp theo
     const nextVersion = await this.getNextVersion(aggregateId);
     // 2. Tao event object
      const event = new Event({
       aggregateId,
       eventType,
       eventData,
       eventVersion: nextVersion,
       timestamp: new Date(),
       metadata: {
          ...metadata,
         source: process.env.SERVICE_NAME || 'unknown-service',
         timestamp: new Date().toISOString()
       }
     });
      // 3. Luu vào MongoDB (Atomic operation)
      const savedEvent = await event.save();
     // 4. Publish lên Kafka
     await this.publishToKafka(eventType, {
       aggregateId,
       eventId: savedEvent._id.toString(),
       eventData,
       eventVersion: nextVersion,
       timestamp: savedEvent.timestamp.toISOString(),
       metadata: savedEvent.metadata
     });
     console.log(`▼ Event written: ${eventType} for ${aggregateId}
(v${nextVersion})`);
      return savedEvent;
   } catch (error) {
      console.error(`X Failed to write event: ${error.message}`);
      throw new Error(`Event write failed: ${error.message}`);
   }
 }
 /**
  * Ghi nhiều events trong một transaction
```

```
async writeEvents(aggregateId, events, metadata = {}) {
  const session = await mongoose.startSession();
  session.startTransaction();
  try {
    const savedEvents = [];
    let currentVersion = await this.getNextVersion(aggregateId);
    for (const eventData of events) {
      const event = new Event({
        aggregateId,
        eventType: eventData.type,
        eventData: eventData.data,
        eventVersion: currentVersion++,
        timestamp: new Date(),
        metadata
      });
      const savedEvent = await event.save({ session });
      savedEvents.push(savedEvent);
    }
    await session.commitTransaction();
    // Publish tất cả events lên Kafka
    for (const event of savedEvents) {
      await this.publishToKafka(event.eventType, {
        aggregateId: event.aggregateId,
        eventId: event._id.toString(),
        eventData: event.eventData,
        eventVersion: event.eventVersion,
        timestamp: event.timestamp.toISOString(),
        metadata: event.metadata
     });
    }
    return savedEvents;
  } catch (error) {
    await session.abortTransaction();
    throw error;
  } finally {
    session.endSession();
  }
}
* Lấy version tiếp theo cho aggregate
*/
async getNextVersion(aggregateId) {
  const lastEvent = await Event
    .findOne({ aggregateId })
    .sort({ eventVersion: -1 })
```

```
.select('eventVersion')
      .lean():
    return lastEvent ? lastEvent.eventVersion + 1 : 1;
  }
  /**
  * Publish event lên Kafka
  async publishToKafka(eventType, eventData) {
    try {
      const topic = this.getTopicForEventType(eventType);
      const partitionKey = eventData.aggregateId; // Đảm bảo ordering
      await this.producer.send({
        topic,
        messages: [{
          key: partitionKey,
          value: JSON.stringify(eventData),
          headers: {
            eventType: eventType,
            aggregateId: eventData.aggregateId,
            timestamp: eventData.timestamp
        }]
      });
      console.log(`♠ Published to Kafka: ${topic}`);
    } catch (error) {
      console.error(`X Kafka publish failed: ${error.message}`);
      // Event đã được lưu trong MongoDB, có thể retry sau
      throw error;
    }
  }
  /**
   * Map event type thanh Kafka topic
  getTopicForEventType(eventType) {
    const topicMap = {
      'UserCreated': 'user.created',
      'UserUpdated': 'user.updated',
      'UserLoggedIn': 'user.logged_in',
      'RegistrationCreated': 'registration.created',
      'NotificationSent': 'notification.sent',
      'AuditLogged': 'audit.logged'
    };
    return topicMap[eventType] || 'unknown.events';
  }
}
module.exports = new EventWriter();
```

### 3.3 Event Reader Service

```
// services/eventReader.js - Đọc sự kiện từ Event Store
const Event = require('../models/eventStore');
class EventReader {
  /**
  * Lấy tất cả events của một aggregate
  * @param {string} aggregateId - ID của aggregate
  * @param {number} fromVersion - Từ version nào (optional)
  */
  async getEventsByAggregate(aggregateId, fromVersion = 0) {
   try {
     const query = { aggregateId };
      if (fromVersion > 0) {
       query.eventVersion = { $gte: fromVersion };
      const events = await Event
        .find(query)
        .sort({ eventVersion: 1 })
        .lean();
      console.log(`\pi Retrieved ${events.length} events for
${aggregateId}`);
      return events;
   } catch (error) {
      console.error(`X Failed to read events: ${error.message}`);
      throw error;
   }
 }
  * Lây events theo loai và thời gian
  * @param {string} eventType - Loai sư kiến
  * @param {Date} fromDate - Từ thời điểm
  * @param {Date} toDate - Đến thời điểm
  */
  async getEventsByTypeAndTime(eventType, fromDate, toDate) {
   const query = { eventType };
   if (fromDate || toDate) {
     query.timestamp = {};
     if (fromDate) query.timestamp.$gte = fromDate;
      if (toDate) query.timestamp.$lte = toDate;
   }
    return await Event
```

```
.find(query)
      .sort({ timestamp: 1 })
      .lean();
  }
  /**
  * Replay events để tái tạo trạng thái aggregate
  * @param {string} aggregateId - ID của aggregate
  async replayEvents(aggregateId) {
    try {
      const events = await this.getEventsByAggregate(aggregateId);
     // Khởi tạo state rỗng
     let aggregateState = {
        id: aggregateId,
       version: 0
      }:
     // Apply từng event để tái tao state
      for (const event of events) {
        aggregateState = this.applyEvent(aggregateState, event);
        aggregateState.version = event.eventVersion;
      console.log(`@ Replayed ${events.length} events for
${aggregateId}`);
      return aggregateState;
    } catch (error) {
      console.error(`X Event replay failed: ${error.message}`);
     throw error;
    }
 }
  * Apply một event lên aggregate state
  applyEvent(currentState, event) {
   const { eventType, eventData } = event;
    switch (eventType) {
      case 'UserCreated':
        return {
          ...currentState,
          username: eventData.username,
          email: eventData.email,
          fullName: eventData.fullName,
          createdAt: eventData.createdAt,
          status: 'active'
        };
      case 'UserUpdated':
        return {
```

```
...currentState,
        ...eventData,
        updatedAt: event.timestamp
      };
    case 'UserLoggedIn':
      return {
        ...currentState,
        lastLoginAt: event.timestamp,
        loginCount: (currentState.loginCount || 0) + 1
      };
    default:
      console.warn(`\( Unknown event type: ${eventType}`);
      return currentState;
 }
}
/**
* Lây event stream cho real-time processing
* @param {string} aggregateId - ID của aggregate (optional)
* @param {number} fromVersion - Từ version (optional)
*/
async getEventStream(aggregateId = null, fromVersion = null) {
  const query = {};
  if (aggregateId) query.aggregateId = aggregateId;
  if (fromVersion) query.eventVersion = { $gte: fromVersion };
  // Sử dụng MongoDB Change Streams cho real-time
  const changeStream = Event.watch([
    { $match: query }
  ]);
 return changeStream;
}
* Tìm kiếm events theo pattern
* @param {object} searchCriteria - Tiêu chí tìm kiếm
*/
async searchEvents(searchCriteria) {
 const {
    eventTypes = [],
    aggregateIds = [],
    fromDate,
    toDate,
    limit = 100,
    offset = 0
  } = searchCriteria;
  const query = {};
  if (eventTypes.length > 0) {
    query.eventType = { $in: eventTypes };
```

```
if (aggregateIds.length > 0) {
      query.aggregateId = { $in: aggregateIds };
    if (fromDate || toDate) {
      query.timestamp = {};
      if (fromDate) guery.timestamp.$qte = fromDate;
      if (toDate) query.timestamp.$lte = toDate;
    }
    return await Event
      .find(query)
      .sort({ timestamp: -1 })
      .skip(offset)
      .limit(limit)
      .lean();
 }
}
module.exports = new EventReader();
```

# → Bước 4: Kafka Producer/Consumer Implementation

#### 4.1 Kafka Producer Service

```
// services/kafkaProducer.js - Kafka Producer để publish events
const { Kafka } = require('kafkajs');
class KafkaProducerService {
  constructor() {
    this.kafka = new Kafka({
      clientId: process.env.SERVICE_NAME || 'event-store-service',
      brokers: (process.env.KAFKA_BROKERS || 'localhost:9092').split(',')
    });
    this.producer = this.kafka.producer({
     maxInFlightRequests: 1,
      idempotent: true,
      transactionTimeout: 30000
    });
   this.isConnected = false;
  }
  async connect() {
    if (!this.isConnected) {
      await this.producer.connect();
      this.isConnected = true;
      console.log('▼ Kafka Producer connected');
```

```
}
}
async disconnect() {
  if (this.isConnected) {
    await this.producer.disconnect();
    this.isConnected = false;
    console.log('▼ Kafka Producer disconnected');
  }
}
/**
* Publish single event
*/
async send(topic, message, partitionKey = null) {
  await this.connect();
  try {
    const result = await this.producer.send({
      topic,
      messages: [{
        key: partitionKey || message.aggregateId,
        value: JSON.stringify(message),
        headers: {
          eventType: message.eventType || '',
          aggregateId: message.aggregateId || '',
          timestamp: new Date().toISOString(),
          source: process.env.SERVICE_NAME || 'unknown'
        }
      }]
    });
    console.log(`♠ Event published to ${topic}:`, result);
    return result;
  } catch (error) {
    console.error(`X Failed to publish to ${topic}:`, error);
    throw error;
  }
}
* Publish multiple events in batch
async sendBatch(topicMessages) {
  await this.connect();
  try {
    const result = await this.producer.sendBatch({
      topicMessages: topicMessages.map(tm => ({
        topic: tm.topic,
        messages: tm.messages.map(msg => ({
          key: msg.key || msg.aggregateId,
          value: JSON.stringify(msg.value),
```

```
headers: {
              eventType: msg.eventType || '',
              aggregateId: msg.aggregateId || '',
              timestamp: new Date().toISOString(),
              source: process.env.SERVICE_NAME || 'unknown'
          }))
       }))
      });
      console.log(' Batch published:', result);
      return result:
    } catch (error) {
      console.error('X Batch publish failed:', error);
      throw error;
   }
 }
}
module.exports = new KafkaProducerService();
```

#### 4.2 Kafka Consumer Service

```
// services/kafkaConsumer.js - Kafka Consumer để consume events
const { Kafka } = require('kafkajs');
class KafkaConsumerService {
  constructor(groupId, topics = []) {
   this.kafka = new Kafka({
      clientId: process.env.SERVICE_NAME || 'event-consumer-service',
      brokers: (process.env.KAFKA_BROKERS || 'localhost:9092').split(',')
    });
    this.consumer = this.kafka.consumer({
      groupId,
      sessionTimeout: 30000,
      heartbeatInterval: 3000,
     maxInFlightRequests: 1 // Ensure ordering
    });
   this.topics = topics;
   this.handlers = new Map();
   this.isRunning = false;
  }
  /**
  * Register event handler
  */
  registerHandler(eventType, handler) {
   this.handlers.set(eventType, handler);
```

```
console.log(`@ Handler registered for: ${eventType}`);
}
/**
* Subscribe to topics and start consuming
async start() {
 await this.consumer.connect();
 // Subscribe to topics
 for (const topic of this.topics) {
   await this.consumer.subscribe({ topic, fromBeginning: false });
   console.log(`\oangle Subscribed to topic: ${topic}`);
 }
 this.isRunning = true;
 // Start consuming
 await this.consumer.run({
    eachMessage: async ({ topic, partition, message, heartbeat }) => {
     try {
        // Parse message
        const eventData = JSON.parse(message.value.toString());
        const eventType = message.headers?.eventType?.toString();
        console.log(` Received event: ${eventType} from ${topic}`);
        // Find and execute handler
        if (this.handlers.has(eventType)) {
          const handler = this.handlers.get(eventType);
          await handler(eventData, { topic, partition, message });
          // Send heartbeat to prevent rebalancing
          await heartbeat();
        } else {
         console.warn(`△ No handler for event type: ${eventType}`);
      } catch (error) {
        console.error(`X Error processing message:`, error);
       // Implement dead letter queue logic here
       await this.handleFailedMessage(message, error);
     }
   }
 });
}
async stop() {
 if (this.isRunning) {
   await this.consumer.disconnect();
   this.isRunning = false;
    console.log('▼ Kafka Consumer stopped');
 }
```

```
}
  /**
  * Handle failed messages (Dead Letter Queue)
  async handleFailedMessage(message, error) {
    // Log to monitoring system
    console.error('@ Dead letter:', {
      topic: message.topic,
      partition: message.partition,
      offset: message.offset,
      error: error.message
    });
    // Could send to dead letter topic
    // await this.sendToDeadLetterQueue(message, error);
  }
}
module.exports = KafkaConsumerService;
```

### 4.3 Event Handlers Implementation

```
// handlers/userEventHandlers.js - Các handler xử lý User events
const eventWriter = require('../services/eventWriter');
const eventReader = require('../services/eventReader');
class UserEventHandlers {
  /**
  * Handler cho UserCreated event
  static async handleUserCreated(eventData, context) {
      console.log(`_ Processing UserCreated: ${eventData.aggregateId}`);
      // Ghi audit log
      await eventWriter.writeEvent(
        eventData.aggregateId,
        'AuditLogged',
          action: 'USER_CREATED',
          userId: eventData.aggregateId,
          details: {
            username: eventData.eventData.username,
            email: eventData.eventData.email
          },
          source: 'user-service',
          timestamp: new Date().toISOString()
        },
```

```
correlationId: eventData.metadata?.correlationId,
          requestId: eventData.metadata?.requestId
        }
      );
      // Trigger welcome email
      await eventWriter.writeEvent(
        `notification ${eventData.aggregateId}`,
        'NotificationTriggered',
          type: 'WELCOME_EMAIL',
          userId: eventData.aggregateId,
          email: eventData.eventData.email,
          templateData: {
            username: eventData.eventData.username,
            welcomeMessage: 'Chào mừng bạn đến với hệ thống!'
          }
        }
      );
      console.log(`\overline{\sigma} UserCreated processed successfully`);
    } catch (error) {
      console.error(`X Failed to process UserCreated:`, error);
      throw error;
    }
  }
  * Handler cho UserLoggedIn event
 static async handleUserLoggedIn(eventData, context) {
    try {
      console.log(`@ Processing UserLoggedIn: ${eventData.aggregateId}`);
      // Update user login statistics
      const userState = await
eventReader.replayEvents(eventData.aggregateId);
      // Ghi audit log
      await eventWriter.writeEvent(
        eventData.aggregateId,
        'AuditLogged',
          action: 'USER_LOGIN',
          userId: eventData.aggregateId,
          details: {
            loginTime: eventData.timestamp,
            ipAddress: eventData.eventData.ipAddress,
            userAgent: eventData.eventData.userAgent,
            totalLogins: (userState.loginCount | | 0) + 1
          },
          source: 'user-service'
```

```
console.log(` UserLoggedIn processed successfully`);

catch (error) {
   console.error(` Failed to process UserLoggedIn:`, error);
   throw error;
}

module.exports = UserEventHandlers;
```

# T Bước 4: Sử Dụng Event-Driven trong Services

### 4.1 User Service - Sử Dung Events

```
// user-service/userController.js
const EventPublisher = require('../services/eventPublisher');
class UserController {
 constructor() {
    this.publisher = new EventPublisher();
 }
  async createUser(req, res) {
   try {
     const userData = req.body;
     // Luu user vào database
      const user = await User.create(userData);
     // Publish event để các services khác biết
      await this.publisher.publishEvent('user-events', {
        eventType: 'UserCreated',
        userId: user.id,
        email: user.email,
        name: user.name
     });
      res.json({ success: true, user });
    } catch (error) {
      res.status(500).json({ error: error.message });
   }
 }
 async updateUser(req, res) {
   try {
     const { id } = req.params;
     const updates = req.body;
```

```
const user = await User.findByIdAndUpdate(id, updates, { new: true
});

// Publish event
await this.publisher.publishEvent('user-events', {
    eventType: 'UserUpdated',
    userId: user.id,
    changes: updates
});

res.json({ success: true, user });
} catch (error) {
    res.status(500).json({ error: error.message });
}
}
}
```

## 4.2 Notification Service - Lắng Nghe Events

```
// notification-service/eventHandlers.js
const EventConsumer = require('../services/eventConsumer');
class NotificationEventHandlers {
 constructor() {
   this.consumer = new EventConsumer('notification-group');
   this.setupEventHandlers();
 }
 async setupEventHandlers() {
    await this.consumer.subscribe(['user-events'],
this.handleEvent.bind(this));
 }
  async handleEvent(topic, event) {
    console.log(`Received event: ${event.eventType}`);
    switch (event.eventType) {
      case 'UserCreated':
        await this.sendWelcomeEmail(event);
        break;
      case 'UserUpdated':
        await this.sendUpdateNotification(event);
        break:
      default:
        console.log(`Unknown event type: ${event.eventType}`);
    }
  }
  async sendWelcomeEmail(event) {
    console.log(`Sending welcome email to: ${event.email}`);
    // Logic gửi email welcome
```

```
async sendUpdateNotification(event) {
   console.log(`User ${event.userId} updated profile`);
   // Logic thông báo update
  }
}
module.exports = NotificationEventHandlers;
```

# Tóm Tắt: Công Cụ và Triển Khai Event-Driven Architecture

# 🛠 Bảng Công Cụ Cần Thiết

Thành Phần	🏋 Công Cụ	Mục Đích
Message Broker	Apache Kafka + Zookeeper	Truyền tải events giữa services
Databases	PostgreSQL + MongoDB	Lưu trữ dữ liệu services
Caching	Redis	Cache và session management
Monitoring	Kafka UI	Quản lý và monitor Kafka
Runtime	Docker + Docker Compose	Container orchestration

## Tac Bước Triển Khai Chính

Bước	<b>⊚</b> Mục Tiêu	Code Chính
1. Setup Infrastructure	Cài đặt Kafka, DB, Redis	docker-compose up
2. Event Publisher	Gửi events qua Kafka	<pre>publishEvent(topic, data)</pre>
3. Event Consumer	Nhận events từ Kafka	<pre>subscribe(topics, handler)</pre>
4. Service Integration	Tích hợp vào business logic	Controller + EventHandlers

# ■ Lợi Ích Event-Driven Architecture

<b>☑</b> Ưu Điểm	✓ Giá Trị
Loose Coupling	Services độc lập, dễ maintain
Scalability	Scale từng service riêng biệt
Resilience	Lỗi 1 service không ảnh hưởng toàn bộ
Real-time	Xử lý events ngay lập tức
Event History	Audit trail tự động

# **⊚** Khi Nào Sử Dụng Event-Driven

✓ Nên Dùng	X Không Nên
Microservices architecture	Simple monolith app
Real-time processing cần	Strong consistency required
Multiple business domains	Small team (<5 developers)
High scalability needs	Quick prototype/MVP

**Key Takeaway**: Event-Driven Architecture phù hợp cho hệ thống phức tạp cần scalability và loose coupling, không dành cho ứng dụng đơn giản. return new EventClass(eventData.eventData, eventData.metadata); }

**/\*\*** 

Map event type thanh event class \*/ static getEventClass(eventType) { const eventMap = {
 'UserCreated': require('../events/UserCreatedEvent'), 'UserUpdated':
 require('../events/UserUpdatedEvent'), 'UserLoggedIn': require('../events/UserLoggedInEvent') };

```
return eventMap[eventType] || class UnknownEvent {};
```

}}

module.exports = AggregateRoot;

```
### **5.2 User Aggregate Implementation**
```javascript
// domain/User.js - User Aggregate
const AggregateRoot = require('./AggregateRoot');
const UserCreatedEvent = require('../events/UserCreatedEvent');
const UserUpdatedEvent = require('../events/UserUpdatedEvent');
const UserLoggedInEvent = require('../events/UserLoggedInEvent');
class User extends AggregateRoot {
  constructor(id) {
    super(id);
    this.username = null;
    this.email = null;
    this.fullName = null;
    this.status = 'inactive';
    this.createdAt = null;
    this.updatedAt = null;
    this.lastLoginAt = null;
    this.loginCount = 0;
  }
  /**
   * Create new user - Business logic
   */
```

```
createUser(userData) {
  // Business rules validation
  if (this.status !== 'inactive') {
   throw new Error('User already exists');
  }
  if (!userData.username || !userData.email) {
   throw new Error('Username and email are required');
  }
  // Record domain event
  this recordEvent(new UserCreatedEvent({
    userId: this.id,
    username: userData.username,
    email: userData.email,
    fullName: userData.fullName,
    createdAt: new Date().toISOString()
 }));
}
/**
* Update user information
*/
updateUser(updateData) {
  if (this.status !== 'active') {
   throw new Error('Cannot update inactive user');
  }
  this.recordEvent(new UserUpdatedEvent({
    userId: this.id,
    changes: updateData,
    updatedAt: new Date().toISOString()
 }));
}
/**
* Record user login
*/
recordLogin(loginData) {
  if (this.status !== 'active') {
    throw new Error('Cannot login inactive user');
  }
  this.recordEvent(new UserLoggedInEvent({
    userId: this.id,
    loginAt: new Date().toISOString(),
    ipAddress: loginData.ipAddress,
    userAgent: loginData.userAgent
  }));
}
// Event Handlers (Apply events to aggregate state)
applyUserCreatedEvent(event) {
```

```
this.username = event.data.username;
    this.email = event.data.email;
    this.fullName = event.data.fullName;
    this.status = 'active';
    this.createdAt = event.data.createdAt;
  }
  applyUserUpdatedEvent(event) {
    Object.assign(this, event.data.changes);
    this.updatedAt = event.data.updatedAt;
  }
  applyUserLoggedInEvent(event) {
    this.lastLoginAt = event.data.loginAt;
    this.loginCount++;
  }
  // Getter methods for queries
  isActive() {
    return this.status === 'active';
  getSnapshot() {
    return {
      id: this.id,
      username: this.username,
      email: this.email,
      fullName: this.fullName,
      status: this.status,
      createdAt: this.createdAt,
      updatedAt: this.updatedAt,
      lastLoginAt: this.lastLoginAt,
      loginCount: this.loginCount,
      version: this version
    };
  }
}
module.exports = User;
```

### 5.3 Domain Events

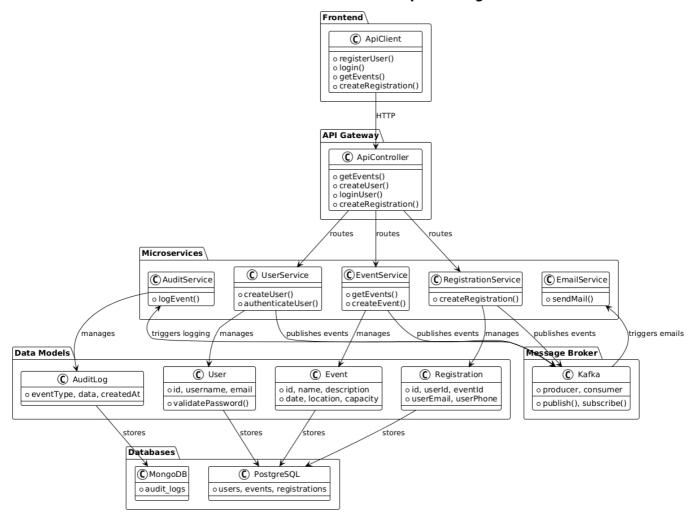
```
// events/UserCreatedEvent.js
class UserCreatedEvent {
  constructor(data, metadata = {}) {
    this.data = data;
    this.metadata = metadata;
    this.timestamp = new Date().toISOString();
    this.eventId = require('crypto').randomUUID();
}
```

```
getEventType() {
    return 'UserCreated';
  getData() {
    return this.data;
  }
}
module.exports = UserCreatedEvent;
// events/UserUpdatedEvent.js
class UserUpdatedEvent {
  constructor(data, metadata = {}) {
    this.data = data;
    this.metadata = metadata;
    this.timestamp = new Date().toISOString();
    this.eventId = require('crypto').randomUUID();
  }
  getEventType() {
    return 'UserUpdated';
}
module.exports = UserUpdatedEvent;
// events/UserLoggedInEvent.js
class UserLoggedInEvent {
  constructor(data, metadata = {}) {
    this.data = data;
    this.metadata = metadata;
    this.timestamp = new Date().toISOString();
    this.eventId = require('crypto').randomUUID();
  }
  getEventType() {
    return 'UserLoggedIn';
  }
}
module.exports = UserLoggedInEvent;
```

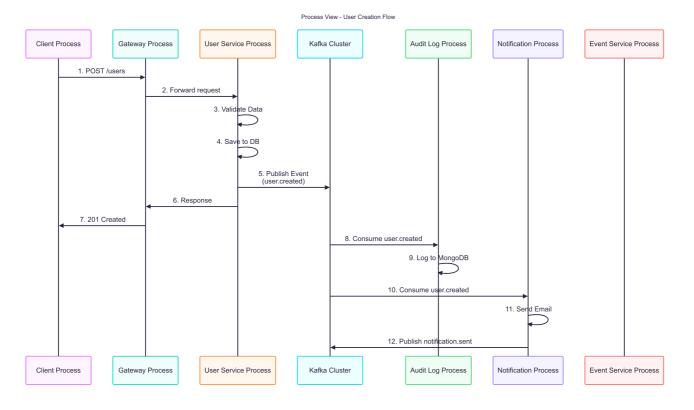
# 

Logical View - Kiến Trúc Logic

### **Event-Driven Architecture Demo - Simplified Logical View**



# Process View - Luồng Xử Lý



#### 1. Input Validation Service

```
// Tools: Joi, Express-validator, Fastify schema validation
// Bước implementation:
1. Schema Definition:
  // schemas/userSchema.js
   const Joi = require('joi');
   const createUserSchema = Joi.object({
     username: Joi.string().alphanum().min(3).max(30).required(),
     email: Joi.string().email().required(),
     password: Joi.string().min(8).pattern(new RegExp('^(?=.*[a-z])(?=.*
[A-Z])(?=.*[0-9])(?=.*[!@\#\s^{\c})')).required(),
     fullName: Joi.string().min(2).max(100).required()
   });
2. Validation Middleware:
   // middleware/validation.is
   const validateInput = (schema) => {
     return (req, res, next) => {
       const { error, value } = schema.validate(req.body);
       if (error) {
         return res.status(400).json({
           success: false,
           message: 'Validation failed',
           errors: error.details.map(detail => ({
             field: detail.path[0],
             message: detail.message
           }))
         });
       }
       req.validatedData = value;
       next();
     };
   };
3. Business Logic Validation:
   // services/userValidationService.js
   class UserValidationService {
     async validateUniqueConstraints(userData) {
       const existingUser = await User.findOne({
           { email: userData.email },
           { username: userData.username }
       });
       if (existingUser) {
         throw new ValidationError('User with this email or username
already exists');
       }
```

```
async validateBusinessRules(userData) {
       // Check age requirements
       if (userData.dateOfBirth) {
        const age = this.calculateAge(userData.dateOfBirth);
         if (age < 13) {
          throw new ValidationError('User must be at least 13 years
old'):
        }
       }
       // Check domain whitelist for email
       const allowedDomains = ['gmail.com', 'company.com'];
       const emailDomain = userData.email.split('@')[1];
       if (!allowedDomains.includes(emailDomain)) {
        throw new ValidationError('Email domain not allowed');
       }
    }
   }
```

#### 2. Event-Driven Data Persistence

```
// Tools: Domain Events, Event Store, CQRS
// Bước implementation:
1. Command Handler với Validation:
   // handlers/CreateUserCommandHandler.js
   class CreateUserCommandHandler {
     constructor(userRepository, validationService, eventBus) {
       this.userRepository = userRepository;
       this.validationService = validationService;
       this.eventBus = eventBus;
     async handle(command) {
       try {
         // 1. Input Validation
         await this.validationService.validateInput(command.userData);
         // 2. Business Rule Validation
this.validationService.validateUniqueConstraints(command.userData);
         await
this.validationService.validateBusinessRules(command.userData);
         // 3. Create Domain Object
         const user = new User(command.userData);
         // 4. Generate Domain Events
         user.recordEvent(new UserCreatedEvent({
```

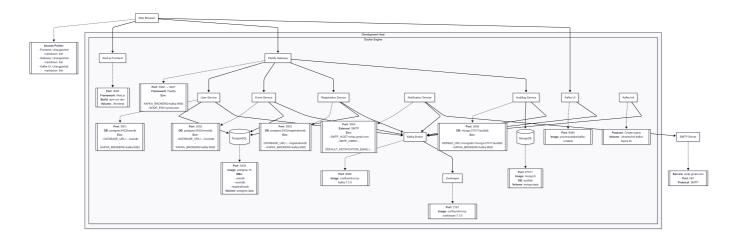
```
userId: user.id,
           username: user.username,
           email: user.email,
           timestamp: new Date().toISOString()
         }));
         // 5. Persist to Database
         await this.userRepository.save(user);
         // 6. Publish Events
         await this.eventBus.publishAll(user.getUncommittedEvents());
         user.markEventsAsCommitted();
         return {
           success: true,
           userId: user.id,
           message: 'User created successfully'
         };
       } catch (error) {
         // 7. Error Handling & Compensation
         await this.handleError(error, command);
         throw error;
       }
     }
     async handleError(error, command) {
       // Log error event
       await this.eventBus.publish(new UserCreationFailedEvent({
         commandId: command.id,
         userData: command.userData,
         error: error.message,
         timestamp: new Date().toISOString()
      }));
   }
2. Event-Driven Repository:
   // repositories/EventDrivenUserRepository.js
   class EventDrivenUserRepository {
     constructor(eventStore, snapshotStore) {
       this.eventStore = eventStore;
      this.snapshotStore = snapshotStore;
     }
     async save(user) {
       const events = user.getUncommittedEvents();
       // Start transaction
       const session = await this.eventStore.startSession();
       session.startTransaction();
       try {
         // Save events atomically
```

```
for (const event of events) {
           await this.eventStore.append({
             aggregateId: user.id,
             eventType: event.constructor.name,
             eventData: event.getData(),
             expectedVersion: user.version,
             timestamp: event.timestamp
           }, { session });
         // Update snapshot if needed
         if (events.length > 10) {
           await this.snapshotStore.save({
             aggregateId: user.id,
             data: user.getSnapshot(),
             version: user.version
           }, { session });
         }
         await session.commitTransaction();
       } catch (error) {
         await session.abortTransaction();
         throw error;
       } finally {
         session.endSession();
       }
     }
     async findById(userId) {
       // Try to load from snapshot first
       const snapshot = await this.snapshotStore.findLatest(userId);
       let user;
       let fromVersion = 0;
       if (snapshot) {
         user = User.fromSnapshot(snapshot.data);
         fromVersion = snapshot.version + 1;
       } else {
         user = new User({ id: userId });
       }
       // Load events since snapshot
       const events = await this.eventStore.getEvents(userId,
fromVersion);
       user.replayEvents(events);
       return user;
     }
   }
3. Complete Feature Implementation:
   // controllers/UserController.js
   class UserController {
```

```
constructor(commandBus, queryBus) {
    this.commandBus = commandBus;
    this queryBus = queryBus;
  async createUser(req, res) {
    try {
     const command = new CreateUserCommand({
        id: generateId(),
        userData: req.validatedData,
        requestId: req.headers['x-request-id'],
        userId: reg.user?.id // For audit trail
      });
      const result = await this.commandBus.send(command);
      res.status(201).json({
        success: true,
        data: {
          userId: result.userId,
          message: result.message
        },
        meta: {
          requestId: command.requestId,
          timestamp: new Date().toISOString()
        }
      });
    } catch (error) {
      if (error instanceof ValidationError) {
        res.status(400).json({
          success: false,
          error: 'Validation failed',
          details: error.message,
          requestId: req.headers['x-request-id']
        });
      } else {
        res.status(500).json({
          success: false,
          error: 'Internal server error',
          requestId: req.headers['x-request-id']
        });
     }
   }
 }
}
```

# Câu 16: Góc Nhìn Triển Khai (Deployment View)

## Deployment Architecture



## X Công Cụ Triển Khai

#### 1. Container Orchestration

```
# Tools: Docker, Kubernetes, Docker Compose
# Bước triển khai:
1. Containerization:
   # Dockerfile cho từng service
   FROM node:18-alpine
   WORKDIR /app
   COPY package*.json ./
   RUN npm ci --only=production
   COPY . .
   EXPOSE 3001
   HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
     CMD curl -f http://localhost:3001/health || exit 1
   CMD ["node", "src/index.js"]
2. Docker Compose (Development):
   # docker-compose.yml
   version: '3.8'
   services:
     user-service:
       build: ./user-service
       ports:
         - "3001:3001"
       environment:
         - DATABASE_URL=postgres://postgres:admin@postgres:5432/userdb
         - KAFKA_BROKERS=kafka:9092
       depends_on:
         - postgres
         kafka
       deploy:
         replicas: 2
         resources:
           limits:
             memory: 512M
             cpus: "0.5"
```

```
3. Kubernetes Deployment:
   # k8s/user-service-deployment.yaml
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: user-service
   spec:
     replicas: 3
     selector:
       matchLabels:
         app: user-service
     template:
       metadata:
         labels:
           app: user-service
       spec:
         containers:
         - name: user-service
           image: your-registry/user-service:latest
           ports:
           - containerPort: 3001
           env:
           - name: DATABASE_URL
             valueFrom:
               secretKeyRef:
                 name: db-secret
                 key: database-url
           resources:
             requests:
               memory: "256Mi"
               cpu: "250m"
             limits:
               memory: "512Mi"
               cpu: "500m"
           livenessProbe:
             httpGet:
               path: /health
               port: 3001
             initialDelaySeconds: 30
             periodSeconds: 10
           readinessProbe:
             httpGet:
               path: /ready
               port: 3001
             initialDelaySeconds: 5
             periodSeconds: 5
```

### 2. Infrastructure as Code

```
# Tools: Terraform, AWS CloudFormation, Ansible
# Bước triển khai:
1. Terraform Infrastructure:
   # infrastructure/main.tf
   provider "aws" {
    region = "us-west-2"
   # EKS Cluster
   module "eks" {
    source = "terraform-aws-modules/eks/aws"
    cluster_name = "eda-demo-cluster"
     cluster version = "1.27"
    vpc_id = module.vpc.vpc_id
     subnet ids = module.vpc.private subnets
    eks_managed_node_groups = {
      main = {
        desired_size = 3
        max\_size = 10
        min_size
                    = 3
        instance_types = ["t3.medium"]
        k8s labels = {
          Environment = "production"
          Application = "eda-demo"
        }
      }
    }
   }
   # RDS PostgreSQL
   resource "aws_db_instance" "postgres" {
     identifier = "eda-demo-postgres"
    engine = "postgres"
     engine_version = "14.7"
    instance_class = "db.t3.micro"
    allocated_storage = 20
    max_allocated_storage = 100
    db_name = "userdb"
     username = "postgres"
     password = var.db_password
     backup_retention_period = 7
     backup_window = "07:00-09:00"
    maintenance_window = "Sun:09:00-Sun:11:00"
```

```
skip_final_snapshot = true
  }
  # MSK Kafka Cluster
  resource "aws_msk_cluster" "kafka" {
     cluster_name
                          = "eda-demo-kafka"
                          = "2.8.1"
    kafka version
    number_of_broker_nodes = 3
    broker_node_group_info {
      instance_type = "kafka.t3.small"
      ebs_volume_size = 20
      client_subnets = module.vpc.private_subnets
      security_groups = [aws_security_group.kafka.id]
    }
  }
2. Ansible Playbook:
  # playbooks/deploy.yml
  name: Deploy EDA Demo Application
    hosts: kubernetes
    tasks:
      name: Apply Kubernetes manifests
           state: present
          definition: "{{ item }}"
        with_fileglob:
          - "../k8s/*.yaml"
      name: Wait for deployment rollout
        k8s_info:
           api_version: apps/v1
          kind: Deployment
          name: "{{ item }}"
          namespace: default
          wait_condition:
            type: Progressing
            status: "True"
            reason: NewReplicaSetAvailable
          wait_timeout: 600
         loop:
          user-service
           - event-service
           registration-service
          - notification-service
           - audit-service
           gateway-service
```

#### 3. CI/CD Pipeline

```
# Tools: GitHub Actions, Jenkins, GitLab CI
# Bước triển khai:
1. GitHub Actions Workflow:
   # .qithub/workflows/deploy.yml
   name: Deploy to Production
   on:
     push:
       branches: [main]
   jobs:
     test:
       runs-on: ubuntu-latest
         - uses: actions/checkout@v3
         - uses: actions/setup-node@v3
             node-version: '18'
         - run: npm ci
         - run: npm run test
         - run: npm run lint
     build:
       needs: test
       runs-on: ubuntu-latest
       strategy:
         matrix:
           service: [user-service, event-service, registration-service,
notification-service, audit-service, gateway]
       steps:
         - uses: actions/checkout@v3
         - name: Build Docker Image
           run:
             docker build -t ${{ secrets.REGISTRY }}/${{ matrix.service}
}}:${{ github.sha }} ./${{ matrix.service }}
             docker build -t ${{ secrets.REGISTRY }}/${{ matrix.service}
}}:latest ./${{ matrix.service }}
         - name: Push to Registry
           run:
             echo ${{ secrets.REGISTRY_PASSWORD }} | docker login ${{
secrets.REGISTRY }} -u ${{ secrets.REGISTRY_USERNAME }} --password-stdin
             docker push ${{ secrets.REGISTRY }}/${{ matrix.service }}:${{
qithub.sha }}
             docker push ${{ secrets.REGISTRY }}/${{ matrix.service}
}}:latest
     deploy:
       needs: build
       runs-on: ubuntu-latest
```

```
environment: production
         - uses: actions/checkout@v3
         - name: Setup kubectl
           uses: azure/setup-kubectl@v3
           with:
             version: 'latest'
         - name: Configure AWS credentials
           uses: aws-actions/configure-aws-credentials@v2
           with:
             aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
             aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
             aws-region: us-west-2
         name: Update kubeconfig
           run: aws eks update-kubeconfig --name eda-demo-cluster
         - name: Deploy to Kubernetes
           run:
             # Update image tags in manifests
             sed -i 's|image: .*|image: ${{ secrets.REGISTRY }}/user-
service:${{ github.sha }}|' k8s/user-service-deployment.yaml
             # Apply manifests
             kubectl apply -f k8s/
             # Wait for rollout
             kubectl rollout status deployment/user-service
             kubectl rollout status deployment/event-service
             kubectl rollout status deployment/registration-service
             kubectl rollout status deployment/notification-service
             kubectl rollout status deployment/audit-service
             kubectl rollout status deployment/gateway-service
2. Monitoring và Logging:
   # monitoring/prometheus-config.yml
   qlobal:
     scrape_interval: 15s
   scrape_configs:
     - job_name: 'kubernetes-services'
       kubernetes_sd_configs:
         - role: service
       relabel_configs:
         - source_labels:
[__meta_kubernetes_service_annotation_prometheus_io_scrape]
           action: keep
           regex: true
         - source_labels:
[__meta_kubernetes_service_annotation_prometheus_io_path]
           action: replace
           target_label: __metrics_path__
```

```
regex: (.+)
   # logging/fluentd-config.yml
   <source>
     @type tail
     path /var/log/containers/*.log
     pos_file /var/log/fluentd-containers.log.pos
     tag kubernetes.*
     format json
   </source>
   <match kubernetes.**>
     @type elasticsearch
     host elasticsearch.logging.svc.cluster.local
     port 9200
     index name kubernetes
   </match>
3. Production Deployment Script:
   #!/bin/bash
   # scripts/deploy-production.sh
   set -e
   echo "# Starting production deployment..."
   # Pre-deployment checks
   echo " Running pre-deployment checks..."
   kubectl get nodes
   kubectl get pods -n default
   # Database migrations
   echo " Running database migrations..."
   kubectl apply -f k8s/migrations-job.yaml
   kubectl wait --for=condition=complete job/db-migration --timeout=300s
   # Deploy services in order
   echo " Deploying services..."
   # 1. Deploy data tier
   kubectl apply -f k8s/postgres-deployment.yaml
   kubectl apply -f k8s/mongodb-deployment.yaml
   kubectl wait --for=condition=ready pod -l app=postgres --timeout=300s
   kubectl wait --for=condition=ready pod -l app=mongodb --timeout=300s
   # 2. Deploy message broker
   kubectl apply -f k8s/kafka-deployment.yaml
   kubectl wait --for=condition=ready pod -l app=kafka --timeout=300s
   # 3. Deploy microservices
   for service in user-service event-service registration-service
notification-service audit-service; do
     echo "Deploying $service..."
     kubectl apply -f k8s/$service-deployment.yaml
```

```
kubectl rollout status deployment/$service --timeout=300s
   done
   # 4. Deploy gateway last
   kubectl apply -f k8s/gateway-deployment.yaml
   kubectl rollout status deployment/gateway-service --timeout=300s
  # Post-deployment verification
   echo "✓ Running post-deployment verification..."
   # Health checks
   for service in user-service event-service registration-service
notification-service audit-service gateway-service; do
    kubectl exec deployment/$service -- curl -f
http://localhost:3001/health
  done
  # Smoke tests
   kubectl apply -f k8s/smoke-tests-job.yaml
   kubectl wait --for=condition=complete job/smoke-tests --timeout=300s
   echo " Production deployment completed successfully!"
   # Display service URLs
   echo " <sup>♀</sup> Service endpoints:"
   kubectl get ingress
```

### Monitoring và Observability

```
# Tools: Prometheus, Grafana, ELK Stack, Jaeger
# Bước setup:
1. Monitoring Stack:
   # monitoring/prometheus-operator.yaml
   apiVersion: monitoring.coreos.com/v1
   kind: ServiceMonitor
   metadata:
     name: eda-demo-services
   spec:
     selector:
       matchLabels:
         monitoring: enabled
     endpoints:
     - port: metrics
       path: /metrics
       interval: 30s
2. Grafana Dashboards:
   # monitoring/grafana-dashboard.json
     "dashboard": {
```

```
"title": "EDA Demo - System Overview",
       "panels": [
         {
           "title": "Event Processing Rate",
           "type": "graph",
           "targets": [
               "expr": "rate(kafka consumer records consumed total[5m])",
               "legendFormat": "{{ service }}"
           1
         },
           "title": "Service Response Times",
           "type": "graph",
           "targets": [
             {
               "expr": "histogram quantile(0.95,
rate(http_request_duration_seconds_bucket[5m]))",
               "legendFormat": "95th percentile - {{ service }}"
           1
         }
     }
   }
3. Distributed Tracing:
   # tracing/jaeger-deployment.yaml
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: jaeger
   spec:
     template:
       spec:
         containers:
         - name: jaeger
           image: jaegertracing/all-in-one:1.35
           ports:
           - containerPort: 16686
           - containerPort: 14268
           - name: COLLECTOR_ZIPKIN_HTTP_PORT
             value: "9411"
```

Hệ thống này đảm bảo:

- V High Availability với multiple replicas
- Scalability với horizontal pod autoscaling
- **Monitoring** với Prometheus/Grafana
- V Logging với ELK stack

- **Tracing** với Jaeger
- Security với network policies và secrets management
- **V** Automation với CI/CD pipelines