# **Online Movie Ticket Booking System**

## **Event-Driven Microservices Architecture on AWS**

# **1. Executive Summary**

The project aims to build a modern online movie ticketing system using event-driven microservices architecture on AWS platform. The system will solve the problems of scalability, reliability and performance for movie theaters.

## **Solution Overview**

The system consists of 6 main microservices: User Service, Mail Service, Showtime Service, Upload Service, Order Service, and Payment Service. The services communicate via message queue (SQS) and API Gateway (Kong gateway), ensuring loosely coupled and independent scalability.

## **Business Benefits**

* Increase order processing capacity by 40% during peak hours
* Reduce response time by 60% compared to monolithic systems
* Save 30% on operating costs with auto-scaling
* Improve customer experience with real-time ticketing system

## **Investment Required**

* Implementation Cost: $2,500 USD (one-time)
* Operating Cost: $180 USD/month
* Time: 4 months of development
* Expected ROI: 250% in 18 months

## **Success Metrics**

* 99.9% uptime availability
* < 200ms response time cho API
* Xử lý 10,000+ concurrent users
* Zero-downtime deployments

# **2. Problem Statement**

## **Current Situation**

Current cinemas are facing serious challenges in managing online ticketing systems:

### **Old Monolithic System**

* **Difficulty in scaling:** Cannot scale individual components
* **Single point of failure:** One error can crash the entire system
* **Long deployment time:** Each update requires deploying the entire application
* **Difficult to maintain:** Large code base, difficult to debug and fix bugs

### **Specific Issues**

1. **Low performance during peak hours**
   * Page load time: 5-10 seconds
   * Timeout rate: 15-20%
   * Lost revenue of approximately $50,000/month due to customer churn
2. **Poor user experience**
   * No real-time notification of seat status
   * Conflicts when multiple people book the same seat
   * Complicated payment process, 35% abandonment rate
3. **Difficulty in operation**
   * No detailed monitoring
   * Difficult to determine the cause of the error
   * Long recovery time (2-4 hours)

## **Impact on Stakeholders**

### **Client**

* Poor booking experience, time consuming
* No trust in the system
* Switch to competitors

### **Business**

* Average lost revenue $600,000/year
* High operating costs due to inefficiency
* Brand reputation affected

### **Technical Team**

* Stress in handling incidents
* Difficulty in developing new features
* Increased work hours

## **Business Consequences**

If not resolved:

* **Lost customers:** 25% of customers may switch to competitors
* **Increased operating costs:** 40% of IT costs due to maintenance
* **Missed opportunities:** Unable to take advantage of peak seasons
* **Legal risks:** Possible violation of SLAs with partners

# **3. Solution Architecture**

## **Architectural Overview**

The system uses Event-Driven Microservices architecture on AWS, ensuring high scalability, reliability and maintainability.

### **AWS Services Used**

#### **Core Infrastructure**

* **Amazon EC2**: Host microservices
* **Amazon VPC**: Network isolation and security

#### **Database Layer**

* **Amazon RDS (PostgreSQL)**: Primary database for business data
* **Amazon ElastiCache (Redis)**: Caching layer
* **Amazon S3**: Object storage cho files

#### **Message Queue & Communication**

* **Amazon MSK**: Async communication between services

#### **Monitoring & Security**

* **Amazon CloudWatch**: Monitoring and logging
* **AWS IAM**: Access control
* **AWS Security Groups**: Network security

## **Microservice Details**

### **1. User Service**

**Function:** Manage authentication, authorization, user profile

**Database:** PostgreSQL

**Technology:** Golang

### **2. Mail Service**

**Function:** Send email notifications (booking confirmation, reminders)

**Technology:** Golang

**Message Queue:** MSK

### **3. Showtime Service**

**Function:** Manage showtimes, seats, availability

**Technology:** Golang

**Database:** PostgreSQL

### **4. Upload Service**

**Function:** Process upload files (movie posters, trailers)

**Technology:** Golang

**Storage:** Amazon S3 Message Queue: SQS consumer APIs

### **5. Order Service**

**Function:** Order Management

**Technology:** Golang

**Database:** PostgreSQL

**6. Payment Service**

**Function:** Payment Processing

**Technology:** Golang

**Database:** PostgreSQL

## **Security Architecture**

### **Network Security**

* **VPC with private subnets:** Microservices are only accessible via API Gateway (Kong gateway)
* **Security Groups:** Strict inbound/outbound rules
* **NAT Gateway:** Outbound internet access for private subnets

### **Application Security**

* **JWT Authentication**: Secure API access
* **API Rate Limiting**: Prevent abuse
* **Input validation**: Sanitize all inputs

### **Data Security**

* **RDS encryption**: Database encryption at rest
* **S3 encryption**: Object encryption
* **IAM roles**: Least privilege access
* **Secrets Manager**: Secure credential storage

## **Scalability Design**

### **Horizontal Scaling**

* **Auto Scaling Groups**: EC2 instances scale based on metrics
* **Load Balancer**: Distribute traffic evenly
* **Database Read Replicas**: Scale read operations

### **Caching Strategy**

* **Redis Cluster**: Cache frequently accessed data

### **Performance Optimization**

* **Connection pooling**: Efficient database connections
* **Async processing**: Non-blocking operations
* **Message queues**: Decouple heavy operations

# **3. Technical Implementation**

## **Deployment Phases**

### **Phase 1: Core Infrastructure (Week 1-2)**

**Deliverables:**

* VPC setup với public/private subnets
* RDS PostgreSQL cluster
* ElastiCache Redis cluster
* S3 buckets configuration
* IAM roles và policies

**Technical Requirements:**

* 2x EC2 t2.micro instances
* RDS db.t3.micro instance
* ElastiCache t3.micro node
* 50GB S3 storage

### **Phase 2: Base Services (Week 3-6)**

**Deliverables:**

* User Service with authentication
* Showtime Service withbasic CRUD
* Database schemas and migrations
* API Gateway setup
* Load balancer configuration

**Development Approach:**

* Microservices pattern with Docker containers
* RESTful API design
* Database-per-service pattern
* API-first development

### **Phase 3: Advanced Services (Week 7-10)**

**Deliverables:**

* Order Service with booking logic
* Payment Service with mock payment gateway
* Upload Service with S3 integration
  + SQS queues and message processing
  + Redis caching implementation

**Technical Requirements:**

* SQS standard queues
* SNS topics for notifications
* CloudWatch custom metrics
* Application Load Balancer rules

### **Phase 4: Integration & Testing (Week 11-14)**

**Deliverables:**

* Mail Service with email notifications
* End-to-end integration
* Comprehensive testing suite
* Performance optimization
* Security hardening

**Testing Strategy:**

* **Unit Tests**: 90% code coverage
* **Integration Tests**: API contract testing
* **Load Tests**: 10,000 concurrent users
* **Security Tests**: OWASP compliance

### **Phase 5: Deployment & Monitoring (Week 15-16)**

**Deliverables:**

* Production deployment
* Monitoring dashboards
* Alerting configuration
* Documentation
* Training materials

**Deployment Plan:**

* Blue-green deployment strategy
* Database migration procedures
* Rollback procedures
* Health checks implementation

## **Configuration Management**

### **Environment Management**

* **Development**: Single EC2 instance
* **Staging**: Production-like environment
* **Production**: Multi-AZ deployment

### **Infrastructure as Code**

* **Docker**: Application containerization

# **4. Timeline & Milestones**

## **Project Overview: 16 Weeks**

### **Phase 1: Foundation (Week 1-2)**

**Milestone 1.1: Infrastructure Setup**

* ✅ VPC and networking complete
* ✅ RDS and ElastiCache operational
* ✅ S3 buckets configured
* ✅ IAM roles established

**Success Criteria:**

* All AWS services deployed successfully
* Network connectivity verified
* Security groups configured
* Database connections established

### **Phase 2: Core Services (Week 3-6)**

**Milestone 2.1: User Management**

* ✅ User Service deployed
* ✅ Authentication working
* ✅ API Gateway configured
* ✅ Load balancer active

**Milestone 2.2: Content Management**

* ✅ Showtime Service deployed
* ✅ Movie catalog functional
* ✅ Seat management working
* ✅ Database schemas optimized

**Success Criteria:**

* User registration/login working
* Movie listings displayed
* Seat availability real-time
* API response time < 200ms

### **Phase 3: Business Logic (Week 7-10)**

**Milestone 3.1: Booking System**

* ✅ Order Service deployed
* ✅ Booking flow implemented
* ✅ Payment Service integrated
* ✅ SQS message processing

**Milestone 3.2: File Management**

* ✅ Upload Service deployed
* ✅ S3 integration working
* ✅ Image processing pipeline
* ✅ Async file processing

**Success Criteria:**

* Complete booking flow working
* Payment processing functional
* File uploads successful
* Message queues operational

### **Phase 4: Integration (Week 11-14)**

**Milestone 4.1: Email Notifications**

* ✅ Mail Service deployed
* ✅ Email templates created
* ✅ Notification triggers working
* ✅ SQS consumers operational

**Milestone 4.2: Performance Optimization**

* ✅ Caching implemented
* ✅ Database queries optimized
* ✅ Load testing completed
* ✅ Auto-scaling configured

**Success Criteria:**

* Email notifications working
* System handles 10,000+ concurrent users
* Response time < 200ms
* Auto-scaling triggers working

### **Phase 5: Deployment (Week 15-16)**

**Milestone 5.1: Production Ready**

* ✅ Production deployment
* ✅ Monitoring dashboards
* ✅ Alerting configured
* ✅ Documentation complete

**Success Criteria:**

* Production system stable
* All monitoring metrics green
* Team trained on operations
* Documentation approved

## **Dependencies and Critical Path**

### **External Dependencies**

* **AWS Account Setup**: Week 1
* **Domain Registration**: Week 2
* **SSL Certificate**: Week 3
* **Payment Gateway API**: Week 8

### **Internal Dependencies**

* Infrastructure → Services Development
* User Service → All other services
* Database Schema → Service Implementation
* Testing Environment → Integration Testing

### **Critical Path**

1. Infrastructure Setup (2 Week)
2. User Service Development (2 Week)
3. Core Services Development (4 Week)
4. Integration & Testing (4 Week)
5. Production Deployment (2 Week)

## **Resource Allocation**

### **Development Team**

* **1 Senior Developer**: Architecture & complex services
* **2 Junior Developers**: Service implementation
* **1 DevOps Engineer**: Infrastructure & deployment

### **Weekly Effort Distribution**

* **Tuần 1-2**: 100% Infrastructure
* **Tuần 3-6**: 70% Development, 30% Testing
* **Tuần 7-10**: 80% Development, 20% Testing
* **Tuần 11-14**: 50% Development, 50% Testing
* **Tuần 15-16**: 30% Development, 70% Deployment

# **5. Budget Estimation**

## **AWS Infrastructure Costs (Monthly)**

### **Compute Services**

| Service | Instance Type | Quantity | Monthly Cost |
| --- | --- | --- | --- |
| EC2 (Production) | t3.medium | 2 | $60 |
| EC2 (Development) | t3.micro | 1 | $8 |
| Application Load Balancer |  | 1 | $22 |
| Subtotal Compute |  |  | $90 |

### **Database Services**

| Service | Instance Type | Quantity | Monthly Cost |
| --- | --- | --- | --- |
| RDS PostgreSQL | t3.medium | 1 | $15 |
| ElastiCache Redis | t3.micro | 1 | $12 |
| Subtotal Database |  |  | $27 |

### **Storage & Networking**

| Service | Usage | Monthly Cost |
| --- | --- | --- |
| S3 Standard | 100GB | $3 |
| Subtotal Database |  | $3 |

### **Monitoring & Security**

| Service | Usage | Monthly Cost |
| --- | --- | --- |
| CloudWatch | Standard metrics | $10 |
| SQS | 1M messages | $0.5 |
| SNS | 1M notifications | $0.5 |
| Subtotal Database |  | $11 |

### 

### **Total AWS Cost: $177/month**

### **ROI Analysis**

### **Quantitative Benefits**

1. **Revenue increase:** 15% from better UX = $90,000/year
2. **Operating cost reduction:** 30% = $18,000/year
3. **Efficiency increase:** 25% staff productivity = $30,000/year
4. **Downtime reduction:** 90% = $50,000/year

### **Total Benefits: $188,000/year**

### **Investment: $2,500**

### **ROI: 7,420% in first year**

# **6. Risk Assessment**

## **Risk Matrix**

| Risk | Probability | Impact | Priority | Mitigation |
| --- | --- | --- | --- | --- |
| Technical debt | High | Medium | High | Code reviews, refactoring |
| AWS costs overrun | Medium | High | High | Cost monitoring, alerts |
| Performance issues | Medium | High | High | Load testing, optimization |
| Performance issues | Medium | High | High | Load testing, optimization |
| Security vulnerabilities | Low | High | Medium | Security audits, testing |
| Team knowledge gaps | High | Medium | Medium | Training, documentation |

## **Risk Details**

### **1. Technical Risks**

**Risk: Performance does not meet requirements**

* **Probability**: Medium (40%)
* **Impact**: High - Impact on user experience
* **Mitigation Strategies**:
  + Load testing from the beginning
  + Performance monitoring real-time
  + Auto-scaling configuration
  + Database query optimization
* **Contingency Plan**: Upgrade instances, add caching layers

**Risk: Microservices complexity**

* **Probability**: High (60%)
* **Impact**: Medium - Increase development time
* **Mitigation Strategies**:
  + Proper service boundaries design
  + Comprehensive documentation
  + Service mesh implementation
  + Monitoring và tracing
* **Contingency Plan**: Simplify architecture, merge services

### **2. Business Risks**

**Risk: AWS costs exceed budget**

* **Probability**: Medium (30%)
* **Impact**: High - ROI Impact
* **Mitigation Strategies**:
  + AWS cost monitoring and alerts
  + Regular cost optimization reviews
  + Reserved instances for predictable workloads
  + Proper resource tagging
* **Contingency Plan**: Downgrade instances, optimize usage

**Risk: Long implementation time**

* **Probability**: Medium (40%)
* **Impact**: Medium - Delay business benefits
* **Mitigation Strategies**:
  + Agile development methodology
  + Regular sprint reviews
  + Clear milestone definitions
  + Risk buffer in timeline
* **Contingency Plan**: Reduce scope, parallel development

### **3. Operational Risks**

**Risk: Security vulnerabilities**

* **Probability**: Low (20%)
* **Impact**: High - Data breach, compliance issues
* **Mitigation Strategies**:
  + Security best practices implementation
  + Regular security audits
  + Penetration testing
  + Employee security training
* **Contingency Plan**: Incident response plan, security patches

**Risk: Lack of expertise**

* **Probability**: High (50%)
* **Impact**: Medium - Quality issues, delays
* **Mitigation Strategies**:
  + Comprehensive training program
  + Mentorship from senior developers
  + External consultation
  + Knowledge sharing sessions
* **Contingency Plan**: Hire external consultants, extended training

## **Monitoring and Escalation**

### **Risk Monitoring Procedures**

1. **Weekly risk reviews**: Team meetings
2. **Monthly stakeholder updates**: Progress and risk status
3. **Quarterly risk assessments**: Comprehensive review
4. **Real-time alerts**: Critical system metrics

# **7. Expected Outcomes**

## **Success Metrics**

### **Technical Metrics**

| Metric | Target | Current | Improvement |
| --- | --- | --- | --- |
| API Response Time | < 200ms | 2-5s | 90% faster |
| System Uptime | 99.9% | 95% | 4.9% increase |
| Concurrent Users | 10,000+ | 1,000 | 10x capacity |
| Deployment Time | < 5 minutes | 2 hours | 95% faster |

### **Business Metrics**

| Metric | Target | Current | Improvement |
| --- | --- | --- | --- |
| Booking Success Rate | 98% | 75% | 23% increase |
| User Satisfaction | 4.5/5 | 3.2/5 | 40% increase |
| Revenue per User | $25 | 18 | 39% increase |
| Operational Cost | -30% | Base line | $18K savings |

## **Short Term Benefits (0-6 months)**

### **Technical Improvement**

* **Reduce deployment time by 80%:** From 2 hours to 10 minutes
* **Increase capacity by 50%:** From 1,000 to 5,000 concurrent users
* **Reduce bugs by 70%:** Thanks to automated testing and code quality
* **Real-time monitoring:** Detect issues in 1 minute

### **Improve Business**

* **20% increase in conversion rate:** Improved booking flow
* **40% reduction in support tickets:** More stable system
* **30% increase in customer retention:** Better user experience
* **25% reduction in operational overhead:** Automated processes

## **Medium Term Benefits (6-18 months)**

### **New Power**

* **Mobile app ready**: API-first architecture
* **Multi-cinema support**: Scalable architecture
* **Real-time analytics**: Business intelligence
* **A/B testing capability**: Data-driven decisions

### **Optimization**

* **Auto-scaling perfected**: Cost optimization
* **Machine learning integration**: Recommendation engine
* **Advanced caching**: Sub-100ms response times
* **International expansion**: Multi-region deployment

## **Long Term Benefits (18+ months)**

### **Strategic Value**

* **Technology leadership**: Modern architecture foundation
* **Developer productivity**: 2x faster feature development
* **Business agility**: Quick adaptation to market changes
* **Competitive advantage**: Superior user experience

### **User Impact**

* **Seamless experience**: One-click booking
* **Personalized recommendations**: AI-powered suggestions
* **Omnichannel support**: Web, mobile, kiosk integration
* **Loyalty program**: Integrated rewards system

### **Scalability**

* **New revenue streams**: Premium features, partnerships
* **Geographic expansion**: Multi-region support
* **Service diversification**: Events, concerts booking
* **Data monetization**: Analytics as a service

## **Continuous Improvement**

### **Feedback Loops**

* **User feedback**: Monthly surveys, NPS tracking
* **System metrics**: Real-time monitoring, alerts
* **Business metrics**: Revenue, conversion tracking
* **Team retrospectives**: Process improvement

### **Innovation Pipeline**

* **Quarterly feature releases**: New capabilities
* **Technology upgrades**: Stay current with trends
* **Partnership opportunities**: Third-party integrations
* **R&D investments**: Future technology exploration

# **Appendices**

## **A. Technical Specifications**

## **B. Cost Calculations**

### **AWS Pricing Details**

**EC2 t3.medium (2 instances):**

- On-demand: $0.0416/hour × 24 × 30 × 2 = $59.90/month

- Reserved (1 year): $0.0277/hour × 24 × 30 × 2 = $39.85/month

**RDS db.t3.micro:**

- On-demand: $0.020/hour × 24 × 30 = $14.40/month

- Reserved (1 year): $0.013/hour × 24 × 30 = $9.36/month

ElastiCache t3.micro:

- On-demand: $0.017/hour × 24 × 30 = $12.24/month

**S3 Standard Storage:**

- First 50 TB: $0.023/GB/month

- 100 GB: $2.30/month

**Data Transfer:**

- First 10 TB: $0.09/GB

- 500 GB: $45/month

### **Cost Optimization Strategies**

1. **Reserved Instances**: 40% savings for predictable workloads
2. **Spot Instances**: 70% savings for development environments
3. **S3 Intelligent Tiering**: 30% savings on storage costs
4. **CloudWatch Logs optimization**: Reduce log retention periods
5. **Auto-scaling**: Match capacity with demand

## **C. Architecture Diagrams**

