Curated Events Platform - Technical Architecture Deep Dive

Overall System Architecture

High-Level Architecture Overview

The platform follows a microservices architecture with event-driven communication, designed for scalability, maintainability, and high availability. The system is built on cloud-native principles with containerized services running on Kubernetes.

```
┌─────────────────────────────────────────────────────────────────
│                    CLIENT LAYER                      │
├─────────────────────────────────────────────────────────────────
│ Web App (React)  │  Mobile Apps (React Native)  │  Admin Panel │
└─────────────────────────────────────────────────────────────────
                           │
┌─────────────────────────────────────────────────────────────────
│               API GATEWAY & CDN                      │
├─────────────────────────────────────────────────────────────────
│ AWS CloudFront │ API Gateway │ Load Balancer │ Rate Limiting  │
└─────────────────────────────────────────────────────────────────
                           │
┌─────────────────────────────────────────────────────────────────
│               MICROSERVICES LAYER                    │
├─────────────────────────────────────────────────────────────────
│ User Service │ Event Service │ Curation Service │ Search Service│
│ Social Service │ Payment Service │ Notification Service      │
│ Analytics Service │ Media Service │ Review Service           │
└─────────────────────────────────────────────────────────────────
                           │
┌─────────────────────────────────────────────────────────────────
│               MESSAGE QUEUE & EVENT BUS              │
├─────────────────────────────────────────────────────────────────
│ Apache Kafka │ Redis Pub/Sub │ AWS SQS │ WebSocket Gateway    │
└─────────────────────────────────────────────────────────────────
                           │
┌─────────────────────────────────────────────────────────────────
│               DATA LAYER                             │
├─────────────────────────────────────────────────────────────────
│PostgreSQL │ MongoDB │ Redis Cache │ Elasticsearch │ S3 Storage │
│ML Feature Store │ Data Warehouse │ Time Series DB            │
└─────────────────────────────────────────────────────────────────
```

Core Technology Stack

Frontend:

- Web: React 18+ with TypeScript, Next.js for SSR/SSG
- Mobile: React Native with Expo for cross-platform development
- State Management: Redux Toolkit + RTK Query
- UI: Tailwind CSS with custom design system

Backend:

- Runtime: Node.js with TypeScript
- Framework: Express.js with Helmet, CORS, compression
- API: GraphQL Federation with REST fallbacks
- Authentication: JWT with refresh tokens, OAuth 2.0

Infrastructure:

- Cloud: AWS (primary) with multi-region deployment
- Containers: Docker + Kubernetes (EKS)
- Service Mesh: Istio for inter-service communication
- Monitoring: Prometheus + Grafana + Jaeger tracing

Databases:

- Primary: PostgreSQL 15+ with read replicas
- Document: MongoDB for flexible event metadata
- Cache: Redis Cluster for session/cache management
- Search: Elasticsearch for event discovery
- Analytics: ClickHouse for real-time analytics

Deep Dive: Major Services

1. User Service

Responsibilities:

- User authentication and authorization
- Profile management and preferences

- Social connections and friend networks
- User reputation and trust scores

Technical Implementation:

```typescript
```

```
// User Service Architecture
interface UserService {
  // Core user operations
  authentication: AuthenticationModule;
  profiles: ProfileManagement;
  preferences: UserPreferences;
  social: SocialConnections;
  reputation: ReputationEngine;
}

// Database Schema Design
interface User {
  id: UUID;
  email: string;
  profile: {
    displayName: string;
    avatar: string;
    location: GeoPoint;
    interests: string[];
    verified: boolean;
  };
  auth: {
    passwordHash: string;
    socialLogins: OAuthProvider[];
    twoFactorEnabled: boolean;
    lastLogin: timestamp;
  };
  reputation: {
    score: number;
    reviewsGiven: number;
    eventsAttended: number;
    communityContributions: number;
  };
  preferences: {
    eventTypes: string[];
    maxDistance: number;
    priceRange: [number, number];
    notifications: NotificationSettings;
  };
}
```

Key Features:

- **Multi-factor Authentication**: SMS, email, and authenticator app support
- **Social Login Integration**: Google, Facebook, Apple Sign-In
- **Reputation System**: Algorithmic scoring based on community participation
- **Privacy Controls**: Granular visibility settings for profile information
- **Fraud Detection**: ML-based detection of fake accounts and suspicious behavior

Data Flow:

1. User registration triggers profile creation and preference setup
2. Authentication events logged for security monitoring
3. Reputation updates broadcast to other services via Kafka
4. Profile changes sync to search index for discoverability

Scaling Considerations:

- Horizontal scaling with stateless authentication
- Redis cluster for session management
- Read replicas for profile data queries
- CDN caching for profile images and public data

---

2. Event Service

Responsibilities:

- Event creation, updates, and lifecycle management
- Event metadata and rich content handling
- RSVP and attendance tracking
- Event categorization and tagging

Technical Implementation:

```typescript
```

```typescript
// Event Service Architecture
interface EventService {
  crud: EventCRUD;
  lifecycle: EventLifecycle;
  attendance: AttendanceManagement;
  media: MediaHandling;
  categorization: AutoCategorization;
}

// Event Schema Design
interface Event {
  id: UUID;
  creator: {
    userId: UUID;
    organizationId?: UUID;
    verified: boolean;
  };
  content: {
    title: string;
    description: string;
    shortDescription: string;
    tags: string[];
    category: EventCategory;
    subcategory: string;
  };
  logistics: {
    startTime: ISO8601;
    endTime: ISO8601;
    timezone: string;
    venue: {
      name: string;
      address: Address;
      coordinates: GeoPoint;
      capacity?: number;
    };
    isOnline: boolean;
    streamingUrl?: string;
  };
  ticketing: {
    isFree: boolean;
    price?: Money;
    capacity?: number;
    ticketUrl?: string;
    requiresApproval: boolean;
  };
  media: {
    coverImage: string;
    gallery: string[];
    video?: string;
  };
  curation: {
    status: CurationStatus;
    reviewHistory: CurationReview[];
    qualityScore: number;
    communityFlags: Flag[];
  };
  engagement: {
    views: number;
    interested: number;
    attending: number;
    shares: number;
    saves: number;
  };
}
```

Key Features:

- **Rich Media Support**: Image optimization, video processing, gallery management
- **Geospatial Indexing**: PostGIS for location-based queries and proximity search
- **Event Templates**: Reusable templates for recurring events
- **Bulk Operations**: Import/export for organizations managing multiple events
- **Version Control**: Track changes to events with rollback capabilities

Data Flow:

1. Event creation triggers AI pre-screening via ML pipeline
2. Approved events indexed in Elasticsearch for search
3. RSVP changes update attendance counters and notify related users
4. Event updates broadcast to subscribers via WebSocket
5. Post-event data archived to data warehouse for analytics

Performance Optimizations:

- Event data cached in Redis with smart invalidation

- Image CDN with automatic format optimization (WebP, AVIF)
- Database partitioning by date and geographic region
- Async processing for heavy operations (image processing, AI analysis)

---

3. Curation Service

Responsibilities:

- AI-powered event pre-screening
- Human curator workflow management
- Community review and flagging system
- Quality scoring and reputation tracking

Technical Implementation:

```typescript
```

```
// Curation Service Architecture
interface CurationService {
  aiScreening: AIPreScreening;
  humanReview: CuratorWorkflow;
  communityReview: CommunityModeration;
  qualityScoring: QualityEngine;
  flagging: FlagSystem;
}

// Curation Workflow
interface CurationWorkflow {
  eventId: UUID;
  stages: {
    aiPreScreening: {
      status: 'pending' | 'passed' | 'failed';
      scores: {
        completeness: number;
        contentQuality: number;
        spamProbability: number;
        imageQuality: number;
      };
      flags: string[];
      processedAt: timestamp;
    };
    humanReview: {
      status: 'pending' | 'approved' | 'rejected' | 'needs_revision';
      curatorId: UUID;
      reviewNotes: string;
      qualityScore: number;
      reviewedAt: timestamp;
    };
    communityOversight: {
      flags: CommunityFlag[];
      reviews: PostEventReview[];
      communityScore: number;
    };
  };
  finalStatus: CurationStatus;
  qualityBadges: QualityBadge[];
}
```

AI Pre-Screening Pipeline:

python

```python
# ML Pipeline for Event Quality Assessment
class EventQualityAssessment:
    def __init__(self):
        self.completeness_analyzer = CompletenessAnalyzer()
        self.content_quality_model = ContentQualityModel()
        self.image_quality_model = ImageQualityModel()
        self.spam_detector = SpamDetectionModel()
        self.duplicate_detector = DuplicateDetector()

    def assess_event(self, event_data):
        scores = {
            'completeness': self.completeness_analyzer.score(event_data),
            'content_quality': self.content_quality_model.predict(event_data.description),
            'image_quality': self.image_quality_model.assess(event_data.cover_image),
            'spam_probability': self.spam_detector.predict(event_data),
            'duplicate_risk': self.duplicate_detector.check(event_data)
        }

        overall_score = self.calculate_weighted_score(scores)
        recommendation = self.make_recommendation(scores, overall_score)

        return CurationAssessment(scores, overall_score, recommendation)
```

Human Curator Dashboard:

- Queue management with priority scoring
- Side-by-side event comparison tools
- Historical creator performance data
- Batch approval/rejection workflows
- Quality trend analytics

Community Review System:

- Post-event feedback from verified attendees
- Reputation-weighted community flags
- Gamified quality contributions
- Appeal process for flagged content

Key Features:

- **ML Quality Models**: Continuously trained on curator decisions
- **Curator Load Balancing**: Smart assignment based on expertise and workload
- **Quality Trend Analysis**: Identify declining event quality patterns
- **Automated Appeals**: AI-assisted review of disputed decisions

---

4. Search & Discovery Service

Responsibilities:

- Real-time event search and filtering
- Personalized recommendations
- Trending events and popularity signals
- Location-based discovery

Technical Implementation:

```typescript
```

```
// Search Service Architecture
interface SearchService {
  indexing: ElasticsearchIndexing;
  querying: SearchQueryEngine;
  recommendations: RecommendationEngine;
  trending: TrendingAnalytics;
  personalization: PersonalizationEngine;
}

// Elasticsearch Index Structure
interface EventSearchIndex {
  id: string;
  title: string;
  description: string;
  category: string;
  subcategories: string[];
  tags: string[];
  location: {
    coordinates: [number, number];
    city: string;
    neighborhood: string;
  };
  datetime: {
    start: Date;
    end: Date;
    day_of_week: number;
    time_of_day: 'morning' | 'afternoon' | 'evening' | 'night';
  };
  pricing: {
    is_free: boolean;
    price_range: 'free' | 'low' | 'medium' | 'high';
    exact_price: number;
  };
  quality: {
    curation_score: number;
    community_rating: number;
    creator_reputation: number;
    quality_badges: string[];
  };
  engagement: {
    views: number;
    interested: number;
    attending: number;
    social_signals: number;
  };
  features: {
    requires_tickets: boolean;
    age_restrictions: boolean;
    accessibility_friendly: boolean;
    pet_friendly: boolean;
    outdoor: boolean;
  };
}
```

Search Query Processing:

```typescript
```

```
// Advanced Search Query Builder
class SearchQueryBuilder {
  buildQuery(searchParams: SearchParams): ElasticsearchQuery {
    const query = {
      bool: {
        must: [],
        filter: [],
        should: [],
        boost: 1.0
      }
    };

    // Text search with boosting
    if (searchParams.query) {
      query.bool.must.push({
        multi_match: {
          query: searchParams.query,
          fields: [
            'title^3',          // Title most important
            'description^2',    // Description secondary
            'tags^1.5',         // Tags moderate boost
            'category'          // Category baseline
          ],
          type: 'cross_fields',
          operator: 'and'
        }
      });
    }

    // Geospatial filtering
    if (searchParams.location && searchParams.radius) {
      query.bool.filter.push({
        geo_distance: {
          distance: `${searchParams.radius}km`,
          'location.coordinates': searchParams.location
        }
      });
    }

    // Quality boosting
    query.bool.should.push({
      function_score: {
        functions: [
        {
          field_value_factor: {
            field: 'quality.curation_score',
            factor: 1.5,
            modifier: 'log1p'
          }
        },
        {
          field_value_factor: {
            field: 'engagement.attending',
            factor: 1.2,
            modifier: 'log1p'
          }
        }
      ],
      score_mode: 'multiply',
      boost_mode: 'multiply'
    }
  });

  return query;
 }
}
```

Recommendation Engine:

```python
```

```python
# Collaborative Filtering + Content-Based Recommendations
class EventRecommendationEngine:
  def __init__(self):
    self.collaborative_model = CollaborativeFilteringModel()
    self.content_model = ContentBasedModel()
    self.popularity_model = PopularityModel()
    self.diversity_optimizer = DiversityOptimizer()

  def get_recommendations(self, user_id: str, limit: int = 20):
    # Get different recommendation signals
    collaborative_recs = self.collaborative_model.recommend(user_id, limit * 2)
    content_recs = self.content_model.recommend(user_id, limit * 2)
    trending_recs = self.popularity_model.get_trending(user_location, limit)

    # Hybrid scoring
    hybrid_scores = self.combine_signals(
      collaborative_recs,
      content_recs,
      trending_recs,
      weights={'collaborative': 0.5, 'content': 0.3, 'trending': 0.2}
    )

    # Optimize for diversity
    final_recs = self.diversity_optimizer.optimize(
      hybrid_scores,
      diversity_factors=['category', 'time', 'price_range'],
      limit=limit
    )

    return final_recs
```

Key Features:

- **Semantic Search**: NLP-powered understanding of event descriptions
- **Real-time Indexing**: Events searchable within seconds of approval
- **Faceted Search**: Multi-dimensional filtering with counts
- **Autocomplete**: Intelligent suggestions for search terms
- **Search Analytics**: Track query performance and user behavior

---

5. Social Service

Responsibilities:

- Social media integration and cross-posting
- Friend connections and social graph
- Activity feeds and social signals
- Group event planning and coordination

Technical Implementation:

```typescript
```

```
// Social Service Architecture
interface SocialService {
  socialGraph: SocialGraphManager;
  integrations: SocialMediaIntegrations;
  activityFeed: ActivityFeedService;
  groupPlanning: GroupPlanningService;
  socialSignals: SocialSignalsAggregator;
}

// Social Graph Data Structure
interface SocialGraph {
  userId: UUID;
  connections: {
    friends: UUID[];
    following: UUID[];
    followers: UUID[];
    blocked: UUID[];
  };
  socialAccounts: {
    platform: 'facebook' | 'instagram' | 'twitter' | 'linkedin';
    accountId: string;
    accessToken: string; // encrypted
    permissions: string[];
    lastSync: timestamp;
  }[];
  privacy: {
    showFriends: boolean;
    showAttendance: boolean;
    allowTagging: boolean;
    activityVisibility: 'public' | 'friends' | 'private';
  };
}
```

Social Media Integration Pipeline:

```
typescript
```

```typescript
// Cross-platform posting system
class SocialMediaIntegrator {
  private platforms = {
    instagram: new InstagramAPI(),
    facebook: new FacebookAPI(),
    twitter: new TwitterAPI(),
    linkedin: new LinkedInAPI()
  };

  async crossPost(event: Event, platforms: string[], userId: UUID) {
    const socialContent = await this.generateSocialContent(event);
    const promises = platforms.map(platform =>
      this.postToPlatform(platform, socialContent, event, userId)
    );

    return await Promise.allSettled(promises);
  }

  private async generateSocialContent(event: Event) {
    return {
      shortText: this.generateShortDescription(event),
      hashtags: this.generateHashtags(event),
      image: await this.optimizeImageForPlatform(event.coverImage),
      link: this.generateEventLink(event.id)
    };
  }
}
```

Activity Feed System:

```
typescript
```

```
// Real-time activity feed
interface ActivityFeed {
  generateFeed(userId: UUID, limit: number): Promise<ActivityItem[]>;
  addActivity(activity: ActivityItem): Promise<void>;
  getActivitiesForEvent(eventId: UUID): Promise<ActivityItem[]>;
}

interface ActivityItem {
  id: UUID;
  type: 'event_created' | 'event_attending' | 'event_review' | 'friend_joined';
  userId: UUID;
  eventId?: UUID;
  content: string;
  metadata: Record<string, any>;
  timestamp: Date;
  visibility: 'public' | 'friends' | 'private';
}
```

Group Planning Features:

- Collaborative event wishlists
- Group voting on event choices
- Shared calendars and scheduling
- Group chat integration
- Split payment coordination

---

6. Payment Service

Responsibilities:

- Payment processing for ticketed events
- Subscription management
- Revenue sharing and payouts
- Financial reporting and compliance

Technical Implementation:

```typescript
```

```typescript
// Payment Service Architecture
interface PaymentService {
  processing: PaymentProcessor;
  subscriptions: SubscriptionManager;
  payouts: PayoutManager;
  compliance: ComplianceManager;
  reporting: FinancialReporting;
}

// Payment Processing Flow
class PaymentProcessor {
  private stripe = new Stripe(config.stripeSecretKey);
  private paypal = new PayPalAPI(config.paypalConfig);

  async processEventPayment(paymentIntent: PaymentIntent): Promise<PaymentResult> {
    try {
      // Create payment intent
      const intent = await this.stripe.paymentIntents.create({
        amount: paymentIntent.amount,
        currency: paymentIntent.currency,
        metadata: {
          eventId: paymentIntent.eventId,
          userId: paymentIntent.userId,
          ticketQuantity: paymentIntent.ticketQuantity
        }
      });

      // Store transaction record
      await this.storeTransaction({
        id: intent.id,
        eventId: paymentIntent.eventId,
        userId: paymentIntent.userId,
        amount: paymentIntent.amount,
        status: 'pending',
        processor: 'stripe'
      });

      return { success: true, clientSecret: intent.client_secret };
    } catch (error) {
      await this.handlePaymentError(error, paymentIntent);
      throw error;
    }
  }
}
```

Revenue Sharing Model:

```typescript
// Automated revenue distribution
interface RevenueShare {
  transactionId: string;
  totalAmount: number;
  distribution: {
    platform: number;     // 3-5% platform fee
    creator: number;      // 85-90% to event creator
    curator: number;      // 1-2% to curator (if applicable)
    processing: number;   // Payment processor fees
    tax: number;          // Tax withholding
  };
  payoutSchedule: 'immediate' | 'weekly' | 'monthly';
}
```

Key Features:

- **Multiple Payment Methods**: Credit cards, PayPal, Apple Pay, Google Pay
- **International Support**: Multi-currency with automatic conversion
- **Fraud Prevention**: ML-based fraud detection and risk scoring
- **PCI Compliance**: Secure tokenization and data handling
- **Automated Payouts**: Scheduled payments to event creators

---

7. Analytics & Intelligence Service

Responsibilities:

- Real-time event and user analytics
- Business intelligence and reporting
- Machine learning model training
- Performance monitoring and optimization

Technical Implementation:

```typescript
```

```
// Analytics Service Architecture
interface AnalyticsService {
  eventTracking: EventTrackingService;
  userBehavior: UserBehaviorAnalytics;
  businessIntelligence: BIReporting;
  mlPipeline: MLPipelineManager;
  realTimeMetrics: RealTimeMetricsService;
}

// Event Tracking Schema
interface AnalyticsEvent {
  id: UUID;
  timestamp: ISO8601;
  userId?: UUID;
  sessionId: UUID;
  eventType: string;
  properties: {
    eventId?: UUID;
    category?: string;
    action: string;
    value?: number;
    metadata: Record<string, any>;
  };
  context: {
    userAgent: string;
    ip: string;
    location: GeoPoint;
    referrer?: string;
    platform: 'web' | 'ios' | 'android';
  };
}
```

Real-time Analytics Pipeline:

```
python
```

```python
# Stream processing for real-time analytics
class RealTimeAnalyticsPipeline:
  def __init__(self):
    self.kafka_consumer = KafkaConsumer('analytics-events')
    self.clickhouse_client = ClickHouseClient()
    self.redis_client = RedisClient()

  def process_events(self):
    for message in self.kafka_consumer:
      event = AnalyticsEvent.from_json(message.value)

      # Store raw event
      self.clickhouse_client.insert('raw_events', event)

      # Update real-time counters
      self.update_real_time_metrics(event)

      # Trigger ML feature updates
      if self.should_update_ml_features(event):
        self.trigger_feature_update(event)

  def update_real_time_metrics(self, event):
    # Update Redis counters for dashboards
    key_patterns = [
      f"events:views:{event.properties.eventId}",
      f"users:activity:{event.userId}",
      f"categories:{event.properties.category}:engagement"
    ]

    for pattern in key_patterns:
      self.redis_client.incr(pattern)
      self.redis_client.expire(pattern, 86400)  # 24 hours
```

Business Intelligence Dashboards:

- Event performance metrics and trends
- User engagement and retention analysis
- Revenue analytics and forecasting
- Curation efficiency and quality metrics
- Geographic market analysis

Infrastructure & DevOps

Deployment Architecture

Container Orchestration:

```yaml
# Kubernetes deployment example
apiVersion: apps/v1
kind: Deployment
metadata:
  name: event-service
spec:
  replicas: 3
  selector:
    matchLabels:
      app: event-service
  template:
    spec:
      containers:
      - name: event-service
        image: events-platform/event-service:latest
        ports:
        - containerPort: 3000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: url
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
        livenessProbe:
          httpGet:
            path: /health
            port: 3000
          initialDelaySeconds: 30
          periodSeconds: 10
```

CI/CD Pipeline:

1. **Code Push**: Triggers automated testing suite
2. **Build**: Docker images built and scanned for vulnerabilities
3. **Test**: Unit, integration, and end-to-end tests
4. **Deploy**: Blue-green deployment to staging
5. **Validation**: Automated smoke tests and manual QA
6. **Production**: Gradual rollout with monitoring

Monitoring & Observability:

- **Metrics**: Prometheus for system metrics, custom business metrics
- **Logging**: Centralized logging with ELK stack (Elasticsearch, Logstash, Kibana)
- **Tracing**: Distributed tracing with Jaeger
- **Alerting**: PagerDuty integration for critical issues
- **Performance**: APM with New Relic or DataDog

Security Architecture

Authentication & Authorization:

- JWT tokens with short expiration and refresh mechanism
- OAuth 2.0 integration with major social platforms
- Role-based access control (RBAC) for admin functions
- API rate limiting and DDoS protection

Data Protection:

- Encryption at rest for all sensitive data
- TLS 1.3 for all client-server communication
- Field-level encryption for PII data
- Regular security audits and penetration testing

Compliance:

- GDPR compliance with data portability and deletion
- PCI DSS compliance for payment processing
- SOC 2 Type II certification planning

- Privacy by design principles

This architecture provides a robust, scalable foundation for the curated events platform while maintaining high quality, performance, and security standards. Each service is designed to operate independently while integrating seamlessly through well-defined APIs and event-driven communication.