KYB Tool - Technical Architecture Document

Part 1: System Design, Microservices, and Data Architecture

Document Information

- **Document Type**: Technical Architecture Document
- **Project**: KYB Tool - Enterprise-Grade Know Your Business Platform
- **Version**: 1.0
- **Date**: January 2025
- **Status**: Final Specification
- **Pages**: Part 1 of 3

---

1. System Design Overview

1.1 Architecture Principles

Microservices-First Design

- Each service owns its data and business logic
- Services communicate via well-defined APIs and events
- Independent deployment and scaling capabilities
- Fault isolation and resilience patterns

Event-Driven Architecture

- Asynchronous processing for non-blocking operations
- Event sourcing for complete audit trails

- Real-time data streaming and processing
- Decoupled service communication

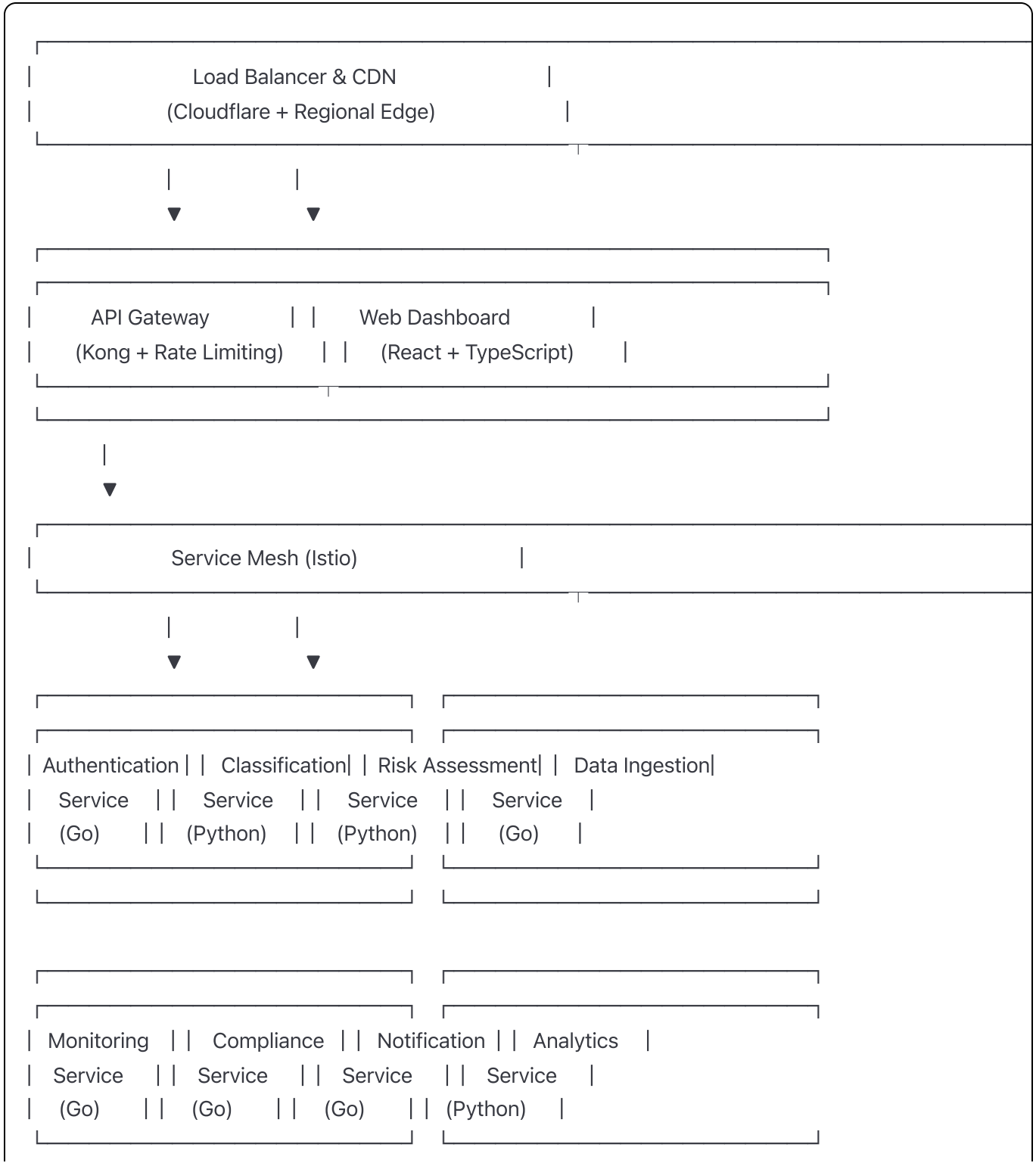
Cloud-Native Design

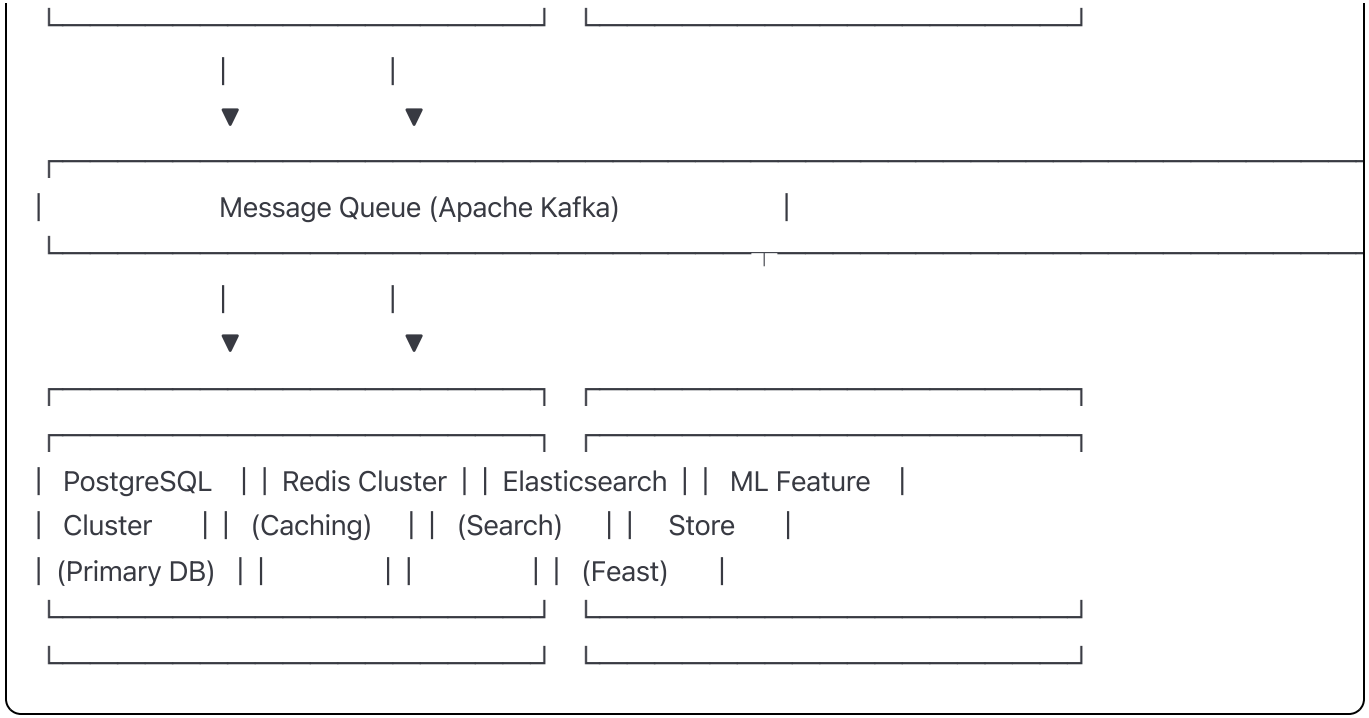
- Container-first deployment strategy
- Kubernetes orchestration for scalability
- Infrastructure as Code (IaC) for consistency
- Multi-region deployment for global scale

Security-by-Design

- Zero-trust network architecture
- End-to-end encryption for all data flows
- Comprehensive audit logging and monitoring
- Compliance-ready security controls

1.2 High-Level System Architecture


Load Balancer & CDN
(Cloudflare + Regional Edge)
API Gateway
(Kong + Rate Limiting)
Web Dashboard
(React + TypeScript)
Service Mesh (Istio)
Authentication Service
(Go)
Classification Service
(Python)
Risk Assessment Service
(Python)
Data Ingestion Service
(Go)
Monitoring Service
(Go)
Compliance Service
(Go)
Notification Service
(Go)
Analytics Service
(Python)



1.3 Technology Stack Selection

Backend Services

- **Go**: High-performance services (API Gateway, Data Ingestion, Monitoring)
- **Python**: AI/ML services (Classification, Risk Assessment, Analytics)
- **Rationale**: Go for I/O intensive operations, Python for ML workloads

Data Layer

- **PostgreSQL**: Primary transactional database with ACID guarantees
- **Redis Cluster**: Distributed caching and session management
- **Elasticsearch**: Full-text search and log aggregation
- **Apache Kafka**: Event streaming and message processing

Infrastructure

- **Kubernetes**: Container orchestration and scaling
- **Istio**: Service mesh for communication and security
- **Prometheus/Grafana**: Monitoring and observability
- **ELK Stack**: Centralized logging and analytics

---

2. Microservices Architecture

2.1 Service Decomposition Strategy

Domain-Driven Design Principles

- Services aligned with business capabilities
- Clear ownership and responsibility boundaries
- Minimal cross-service data dependencies
- Independent evolution and deployment

Service Sizing Guidelines

- 2-pizza team ownership model
- Single responsibility principle
- Database per service pattern
- Independent scaling requirements

2.2 Core Services Definition

Authentication Service (Go)

```yaml
```

Responsibilities:
- User authentication and authorization
- JWT token generation and validation
- API key management and rotation
- Multi-tenant access control
- Session management and SSO preparation

API Endpoints:
- POST /auth/login
- POST /auth/logout
- GET /auth/validate
- POST /auth/refresh
- CRUD /auth/api-keys

Database Schema:
- users (id, tenant_id, email, password_hash, role, created_at, updated_at)
- api_keys (id, tenant_id, key_hash, permissions, rate_limit, created_at, expires_at)
- sessions (id, user_id, token_hash, expires_at, last_accessed)

Dependencies:
- Redis for session storage
- PostgreSQL for user data

Classification Service (Python/FastAPI)

```yaml
```

Responsibilities:
- Business classification using AI models
- MCC, NAICS, SIC code assignment
- Confidence scoring and alternatives
- Website content analysis
- Product/service categorization

API Endpoints:
- POST /classify/business
- POST /classify/batch
- GET /classify/{job_id}/status
- POST /classify/website
- GET /classify/codes/search

ML Models:
- BERT-based business description classifier
- XGBoost ensemble for multi-code prediction
- Custom similarity matching algorithms
- Website content categorization CNN

Data Sources:
- Business registration data
- Website content scraping
- Industry classification databases
- Product/service taxonomies

Risk Assessment Service (Python/FastAPI)

yaml

Responsibilities:
- Real-time risk scoring (1-100 scale)
- Predictive risk modeling (3, 6, 12 months)
- Fraud detection and pattern analysis
- Regulatory compliance risk assessment
- Risk factor explanation and recommendations

API Endpoints:
- POST /risk/assess
- POST /risk/predict
- GET /risk/{merchant_id}/history
- POST /risk/batch-assess
- GET /risk/factors/explain

ML Models:
- XGBoost + LSTM ensemble for risk prediction
- Isolation Forest for anomaly detection
- Graph Neural Networks for transaction analysis
- SHAP for model explainability

Risk Categories:
- Operational Risk (business stability, operational issues)
- Financial Risk (credit, cash flow, financial distress)
- Regulatory Risk (compliance violations, sanctions)
- Reputational Risk (negative news, customer complaints)
- Cybersecurity Risk (data breaches, security incidents)

Data Ingestion Service (Go)

yaml

Responsibilities:
- Web scraping and content extraction
- Third-party API integration management
- Data validation and normalization
- Rate limiting and quota management
- Error handling and retry logic

API Endpoints:
- POST /ingest/website
- POST /ingest/business-registry
- GET /ingest/job/{id}/status
- POST /ingest/validate
- GET /ingest/sources/health

Data Sources:
- Business registry APIs (Secretary of State databases)
- Website content scraping (Playwright-based)
- Social media APIs (when available and free)
- Government databases (OFAC, tax authorities)
- Open data sources (OpenCorporates, etc.)

Features:
- Intelligent bot evasion techniques
- Proxy rotation and IP management
- Content quality assessment
- Duplicate detection and deduplication

Compliance Service (Go)

yaml

Responsibilities:
- Regulatory compliance checking
- Sanctions screening (OFAC, UN, EU)
- Audit trail generation
- Compliance reporting
- Policy enforcement

API Endpoints:
- POST /compliance/screen
- GET /compliance/audit-trail
- POST /compliance/report
- GET /compliance/policies
- POST /compliance/validate

Compliance Frameworks:
- OFAC sanctions screening
- EU sanctions compliance
- PCI DSS data handling
- GDPR privacy protection
- SOC 2 security controls

Features:
- Fuzzy name matching for sanctions screening
- Automated compliance reporting
- Policy violation detection
- Audit log immutability (blockchain-backed)

2.3 Service Communication Patterns

Synchronous Communication

- Direct API calls for real-time operations

- Circuit breaker pattern for fault tolerance
- Timeout and retry policies
- Service mesh for secure communication

Asynchronous Communication

- Event-driven messaging via Kafka
- Event sourcing for audit trails
- Saga pattern for distributed transactions
- Dead letter queues for failed messages

Data Consistency Strategy

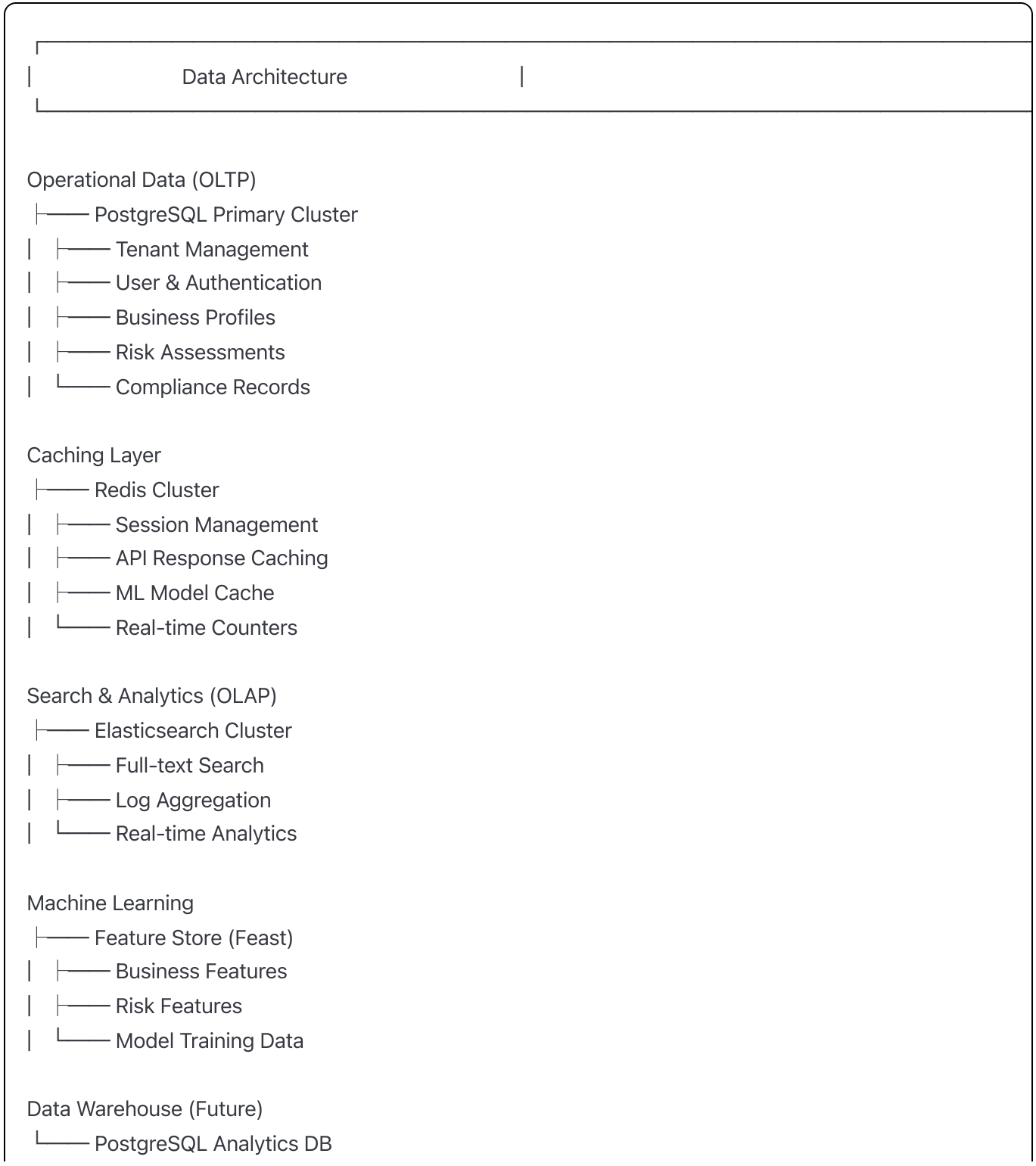
- Eventual consistency between services
- Distributed transaction coordination when required
- Idempotent operations for retry safety
- Conflict resolution strategies

---

3. Database Design and Data Architecture

3.1 Data Architecture Overview

Multi-Store Strategy


Data Architecture
Operational Data (OLTP)
PostgreSQL Primary Cluster
Tenant Management
User & Authentication
Business Profiles
Risk Assessments
Compliance Records
Caching Layer
Redis Cluster
Session Management
API Response Caching
ML Model Cache
Real-time Counters
Search & Analytics (OLAP)
Elasticsearch Cluster
Full-text Search
Log Aggregation
Real-time Analytics
Machine Learning
Feature Store (Feast)
Business Features
Risk Features
Model Training Data
Data Warehouse (Future)
PostgreSQL Analytics DB

```
├── Historical Data
├── Reporting Data
└── Business Intelligence
```

3.2 PostgreSQL Schema Design

Core Business Entities

sql

```sql
-- Tenant Management
CREATE TABLE tenants (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    plan_type VARCHAR(50) NOT NULL,
    api_rate_limit INTEGER DEFAULT 100,
    features JSONB DEFAULT '{}',
    billing_info JSONB,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    deleted_at TIMESTAMP WITH TIME ZONE NULL
);

-- User Management
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    role VARCHAR(50) NOT NULL DEFAULT 'user',
    permissions JSONB DEFAULT '[]',
    last_login TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    deleted_at TIMESTAMP WITH TIME ZONE NULL
);

-- Business Profiles
CREATE TABLE businesses (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    external_id VARCHAR(255), -- Customer's merchant ID
    legal_name VARCHAR(500) NOT NULL,
```

```
    dba_name VARCHAR(500),
    tax_id VARCHAR(50),
    business_type VARCHAR(100),
    registration_state VARCHAR(50),
    registration_country VARCHAR(50),
    registration_date DATE,
    website_url TEXT,
    business_description TEXT,
    contact_info JSONB,
    address JSONB,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

    -- Indexes for performance
    INDEX idx_businesses_tenant_external (tenant_id, external_id),
    INDEX idx_businesses_tax_id (tax_id) WHERE tax_id IS NOT NULL,
    INDEX idx_businesses_website (website_url) WHERE website_url IS NOT NULL
);

-- Business Classifications
CREATE TABLE business_classifications (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    business_id UUID NOT NULL REFERENCES businesses(id) ON DELETE CASCADE,
    classification_type VARCHAR(20) NOT NULL, -- 'MCC', 'NAICS', 'SIC'
    code VARCHAR(20) NOT NULL,
    description TEXT,
    confidence_score DECIMAL(5,4), -- 0.0000 to 1.0000
    is_primary BOOLEAN DEFAULT false,
    source VARCHAR(100), -- 'ai_model', 'manual', 'api'
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

    INDEX idx_classifications_business (business_id),
    INDEX idx_classifications_type_code (classification_type, code)
```

```
);

-- Risk Assessments
CREATE TABLE risk_assessments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    business_id UUID NOT NULL REFERENCES businesses(id) ON DELETE CASCADE,
    assessment_type VARCHAR(50) NOT NULL, -- 'initial', 'periodic', 'triggered'
    overall_score INTEGER NOT NULL CHECK (overall_score BETWEEN 1 AND 100),
    risk_level VARCHAR(20) NOT NULL, -- 'low', 'medium', 'high', 'critical'

    -- Individual risk scores
    operational_risk INTEGER CHECK (operational_risk BETWEEN 1 AND 100),
    financial_risk INTEGER CHECK (financial_risk BETWEEN 1 AND 100),
    regulatory_risk INTEGER CHECK (regulatory_risk BETWEEN 1 AND 100),
    reputational_risk INTEGER CHECK (reputational_risk BETWEEN 1 AND 100),
    cybersecurity_risk INTEGER CHECK (cybersecurity_risk BETWEEN 1 AND 100),

    risk_factors JSONB DEFAULT '[]',
    recommendations JSONB DEFAULT '[]',
    model_version VARCHAR(50),
    confidence_interval JSONB, -- {'lower': 0.8, 'upper': 0.95}

    -- Predictive scores
    risk_trend VARCHAR(20), -- 'increasing', 'stable', 'decreasing'
    predicted_3m INTEGER CHECK (predicted_3m BETWEEN 1 AND 100),
    predicted_6m INTEGER CHECK (predicted_6m BETWEEN 1 AND 100),
    predicted_12m INTEGER CHECK (predicted_12m BETWEEN 1 AND 100),

    assessed_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    valid_until TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

    INDEX idx_risk_assessments_business (business_id),
```

```
    INDEX idx_risk_assessments_score (overall_score),
    INDEX idx_risk_assessments_date (assessed_at)
);

-- Website Analysis
CREATE TABLE website_analyses (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    business_id UUID NOT NULL REFERENCES businesses(id) ON DELETE CASCADE,
    url TEXT NOT NULL,
    content_hash VARCHAR(64), -- SHA-256 of content

    -- Content analysis results
    products_services JSONB DEFAULT '[]',
    prohibited_content JSONB DEFAULT '[]',
    risk_indicators JSONB DEFAULT '[]',
    content_quality_score DECIMAL(5,4),
    ssl_certificate JSONB,

    -- Technical analysis
    page_load_time INTEGER, -- milliseconds
    mobile_friendly BOOLEAN,
    seo_score INTEGER CHECK (seo_score BETWEEN 0 AND 100),

    -- Screenshots and evidence
    screenshot_url TEXT,
    evidence JSONB DEFAULT '{}',

    analyzed_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

    INDEX idx_website_analyses_business (business_id),
    INDEX idx_website_analyses_url_hash (url, content_hash)
);
```

Compliance and Audit Tables

```sql
```

```sql
-- Sanctions Screening
CREATE TABLE sanctions_screenings (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    business_id UUID NOT NULL REFERENCES businesses(id) ON DELETE CASCADE,
    screening_type VARCHAR(50) NOT NULL, -- 'ofac', 'un', 'eu', 'uk'
    entity_name VARCHAR(500) NOT NULL,
    match_found BOOLEAN NOT NULL DEFAULT false,
    match_details JSONB DEFAULT '[]',
    match_score DECIMAL(5,4),
    list_version VARCHAR(100),
    screened_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

    INDEX idx_sanctions_business (business_id),
    INDEX idx_sanctions_match (match_found),
    INDEX idx_sanctions_date (screened_at)
);

-- Audit Trail
CREATE TABLE audit_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    user_id UUID REFERENCES users(id),
    api_key_id UUID, -- For API-driven actions

    event_type VARCHAR(100) NOT NULL, -- 'business_created', 'risk_assessed', etc.
    entity_type VARCHAR(50) NOT NULL, -- 'business', 'user', 'assessment'
    entity_id UUID,

    action VARCHAR(50) NOT NULL, -- 'create', 'update', 'delete', 'read'
    old_values JSONB,
    new_values JSONB,

    ip_address INET,
```

```
    user_agent TEXT,
    request_id UUID,

    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

    INDEX idx_audit_tenant_date (tenant_id, created_at),
    INDEX idx_audit_entity (entity_type, entity_id),
    INDEX idx_audit_user (user_id)
);

-- API Usage Tracking
CREATE TABLE api_usage (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    api_key_id UUID,
    endpoint VARCHAR(200) NOT NULL,
    method VARCHAR(10) NOT NULL,
    status_code INTEGER NOT NULL,
    response_time INTEGER, -- milliseconds
    request_size INTEGER, -- bytes
    response_size INTEGER, -- bytes
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

    INDEX idx_api_usage_tenant_date (tenant_id, created_at),
    INDEX idx_api_usage_endpoint (endpoint)
);
```

3.3 Redis Caching Strategy

Cache Structure and TTL Policies

yaml

Session Management:
  Key Pattern: "session:{token_hash}"
  TTL: 24 hours (configurable)
  Data: User session information, permissions

API Response Caching:
  Key Pattern: "api:{endpoint}:{hash(params)}"
  TTL: 1-24 hours (based on data freshness requirements)
  Data: Serialized API responses

Business Classification Cache:
  Key Pattern: "classify:{hash(business_data)}"
  TTL: 7 days
  Data: Classification results with confidence scores

Risk Assessment Cache:
  Key Pattern: "risk:{business_id}:{assessment_date}"
  TTL: 24 hours
  Data: Risk scores and predictions

Website Analysis Cache:
  Key Pattern: "website:{url_hash}:{content_hash}"
  TTL: 7 days
  Data: Website analysis results

Rate Limiting:
  Key Pattern: "rate_limit:{api_key_id}:{window}"
  TTL: Based on rate limit window (1 minute, 1 hour, 1 day)
  Data: Request counters

Feature Vectors (ML):
  Key Pattern: "features:{business_id}:{feature_set}"

TTL: 1 hour

Data: Preprocessed ML feature vectors

Cache Warming and Invalidation

python

```python
# Intelligent cache warming strategy
class CacheWarmer:
    async def warm_popular_classifications(self):
        """Pre-populate cache with popular business types"""
        popular_queries = await self.get_popular_classification_queries()
        for query in popular_queries:
            await self.classification_service.classify(query, cache_only=False)

    async def warm_risk_models(self):
        """Pre-load ML models into memory/cache"""
        await self.risk_service.load_models()

    async def invalidate_on_update(self, business_id: UUID):
        """Invalidate related cache entries when business data updates"""
        patterns = [
            f"risk:{business_id}:*",
            f"classify:*{business_id}*",
            f"website:*{business_id}*"
        ]
        for pattern in patterns:
            await self.redis.delete_pattern(pattern)
```

3.4 Elasticsearch Schema and Usage

Index Templates for Different Data Types

json

```
{
  "business_profiles": {
    "mappings": {
      "properties": {
        "business_id": {"type": "keyword"},
        "tenant_id": {"type": "keyword"},
        "legal_name": {
          "type": "text",
          "analyzer": "standard",
          "fields": {
            "keyword": {"type": "keyword"}
          }
        },
        "business_description": {
          "type": "text",
          "analyzer": "english"
        },
        "classifications": {
          "type": "nested",
          "properties": {
            "type": {"type": "keyword"},
            "code": {"type": "keyword"},
            "description": {"type": "text"},
            "confidence_score": {"type": "float"}
          }
        },
        "risk_score": {"type": "integer"},
        "risk_level": {"type": "keyword"},
        "website_content": {"type": "text"},
        "created_at": {"type": "date"},
        "@timestamp": {"type": "date"}
      }
    }
```

```
  },

  "audit_logs": {
    "mappings": {
      "properties": {
        "tenant_id": {"type": "keyword"},
        "user_id": {"type": "keyword"},
        "event_type": {"type": "keyword"},
        "entity_type": {"type": "keyword"},
        "action": {"type": "keyword"},
        "ip_address": {"type": "ip"},
        "details": {"type": "object"},
        "@timestamp": {"type": "date"}
      }
    }
  },

  "api_logs": {
    "mappings": {
      "properties": {
        "tenant_id": {"type": "keyword"},
        "endpoint": {"type": "keyword"},
        "method": {"type": "keyword"},
        "status_code": {"type": "integer"},
        "response_time": {"type": "integer"},
        "request_size": {"type": "integer"},
        "response_size": {"type": "integer"},
        "@timestamp": {"type": "date"}
      }
    }
  }
}
```

This completes Part 1 of the Technical Architecture document, covering system design, microservices architecture, and comprehensive data architecture. The document provides specific implementation details for database schemas, caching strategies, and search capabilities.

Should I proceed with **Part 2: AI/ML Architecture, External Integrations, and Security**?