Synthetic Arbitrage Detection Engine - Code Documentation

Software Engineering Team

July 16, 2025

Contents

1 Executive Summary

This code documentation provides comprehensive information about the Synthetic Arbitrage Detection Engine's software architecture, API interfaces, deployment procedures, and maintenance guidelines. The document serves as a reference for developers, system administrators, and DevOps teams.

Key Documentation Sections: - **System Architecture**: High-level design and component interactions - **API Documentation**: RESTful endpoints and WebSocket interfaces - **Setup Instructions**: Development environment configuration - **Deployment Guide**: Production deployment procedures - **Maintenance**: Monitoring, troubleshooting, and updates

Target Audience: - Software Developers - System Administrators - DevOps Engineers - Technical Support Teams - Integration Partners

2 System Architecture Overview

2.1 High-Level Architecture

2.1.1 Component Diagram

```
+----------------------------------------------------------------+
|                        Web Dashboard                         |
|                     (React + TypeScript)                    |
+-------------------------------+--------------------------------+
                               | HTTP/WebSocket
+-------------------------------+--------------------------------+
|                     API Gateway                             |
|                  (NGINX + Load Balancer)                   |
+-------------------------------+--------------------------------+
                               | REST API
+-------------------------------+--------------------------------+
|                   Core Application                          |
|                      (C++20)                               |
+----------------------------------------------------------------+
|  Trading Engine  |  Risk Manager  |  Pricing Engine  |  Monitor |
+-------------------------------+--------------------------------+
                               | Message Queue
+-------------------------------+--------------------------------+
|                   Data Processing Layer                     |
|              (Market Data + Real-time Analytics)           |
+-------------------------------+--------------------------------+
                               | Database Connections
+-------------------------------+--------------------------------+
|                     Data Storage                            |
|          (PostgreSQL + TimescaleDB + Redis)                |
+----------------------------------------------------------------+
```

2.1.2 Technology Stack

Backend Technologies: - **Core Engine**: C++20 with modern features - **Build System**: CMake 3.20+ - **Database**: PostgreSQL 14+ with TimescaleDB - **Caching**: Redis 7.0+ - **Message Queue**: Apache Kafka 3.0+ - **API Framework**: Custom C++ REST framework

Frontend Technologies: - **Dashboard**: React 18+ with TypeScript - **UI Framework**: Material-UI (MUI) - **State Management**: Redux Toolkit - **WebSocket**: Socket.IO client - **Charts**: Chart.js / D3.js

Infrastructure: - **Containerization**: Docker + Docker Compose - **Orchestration**: Kubernetes - **Reverse Proxy**: NGINX - **Monitoring**: Prometheus + Grafana - **Logging**: ELK Stack (Elasticsearch, Logstash, Kibana)

2.2 Core Components

2.2.1 Trading Engine

- **Purpose**: Execute arbitrage strategies and manage positions
- **Language**: C++20
- **Key Features**: Sub-10ms latency, multi-threading, lock-free queues
- **Location**: `src/core/ArbitrageEngine.hpp/cpp`

2.2.2 Risk Manager

- **Purpose**: Monitor and control trading risks
- **Language**: C++20
- **Key Features**: Real-time VaR calculation, position limits
- **Location**: `src/core/RiskManager.hpp/cpp`

2.2.3 Pricing Engine

- **Purpose**: Calculate synthetic instrument prices
- **Language**: C++20
- **Key Features**: Black-Scholes implementation, SIMD optimizations
- **Location**: `src/core/PricingEngine.hpp/cpp`

2.2.4 Market Data Handler

- **Purpose**: Process real-time market data from exchanges
- **Language**: C++20
- **Key Features**: WebSocket connections, data normalization
- **Location**: `src/data/MarketDataHandler.hpp/cpp`

3 API Documentation

3.1 RESTful API Endpoints

3.1.1 Authentication

Base URL: `https://api.arbitrage-engine.com/v1`

Authentication Method: JWT Bearer Token

Request Headers:

```
Authorization: Bearer <jwt_token>
Content-Type: application/json
X-API-Version: 1.0
```

3.1.2 Market Data API

3.1.2.1 Get Current Prices

Endpoint: `GET /market/prices`

Description: Retrieve current market prices for all supported instruments

Parameters: - `symbols` (optional): Comma-separated list of symbols - `exchange` (optional): Filter by exchange name

Response:

```json
{
  "status": "success",
  "timestamp": "2025-07-16T10:30:00Z",
  "data": {
    "prices": [
      {
        "symbol": "BTC/USD",
        "exchange": "binance",
        "price": 45000.50,
        "volume": 1234.56,
        "timestamp": "2025-07-16T10:30:00Z"
      }
    ]
  }
}
```

Error Responses: - `400`: Bad Request - Invalid parameters - `401`: Unauthorized - Invalid token - `404`: Not Found - Symbol not found - `500`: Internal Server Error

3.1.2.2 Get Historical Data

Endpoint: `GET /market/history`

Description: Retrieve historical price data

Parameters: - `symbol` (required): Trading symbol - `interval` (required): Time interval (1m, 5m, 1h, 1d) - `start_time` (required): Start timestamp (ISO 8601) - `end_time` (required): End timestamp (ISO 8601) - `limit` (optional): Maximum number of records (default: 500)

Response:

```json
{
  "status": "success",
  "data": {
    "symbol": "BTC/USD",
    "interval": "1h",
    "records": [
      {
        "timestamp": "2025-07-16T10:00:00Z",
        "open": 44950.00,
        "high": 45100.00,
        "low": 44800.00,
        "close": 45000.50,
        "volume": 2345.67
      }
    ]
  }
}
```

3.1.3 Trading API

3.1.3.1 Get Positions **Endpoint**: `GET /trading/positions`

Description: Retrieve current trading positions

Parameters: - `status` (optional): Filter by status (open, closed, pending) - `symbol` (optional): Filter by symbol

Response:

```json
{
  "status": "success",
  "data": {
    "positions": [
      {
        "id": "pos_123456",
        "symbol": "BTC/USD",
        "type": "long",
        "size": 1.5,
        "entry_price": 44800.00,
        "current_price": 45000.50,
        "unrealized_pnl": 300.75,
        "timestamp": "2025-07-16T09:15:00Z",
        "status": "open"
      }
    ]
  }
}
```

3.1.3.2 Place Order **Endpoint**: `POST /trading/orders`

Description: Place a new trading order

Request Body:

```
{
  "symbol": "BTC/USD",
  "side": "buy",
  "type": "market",
  "quantity": 0.1,
  "price": 45000.00,
  "time_in_force": "GTC"
}
```

Response:

```
{
  "status": "success",
  "data": {
    "order_id": "ord_789012",
    "symbol": "BTC/USD",
    "side": "buy",
    "type": "market",
    "quantity": 0.1,
    "status": "filled",
    "filled_quantity": 0.1,
    "average_price": 45000.50,
    "timestamp": "2025-07-16T10:30:00Z"
  }
}
```

3.1.3.3 Cancel Order **Endpoint**: `DELETE /trading/orders/{order_id}`

Description: Cancel an existing order

Parameters: - `order_id` (path): Order ID to cancel

Response:

```
{
  "status": "success",
  "data": {
    "order_id": "ord_789012",
    "status": "cancelled",
    "timestamp": "2025-07-16T10:31:00Z"
  }
}
```

3.1.4 Risk Management API

3.1.4.1 Get Risk Metrics

Endpoint: `GET /risk/metrics`

Description: Retrieve current risk metrics

Response:

```
{
  "status": "success",
  "data": {
    "portfolio_var": 125000.00,
    "expected_shortfall": 180000.00,
    "max_drawdown": 0.15,
    "leverage_ratio": 3.2,
    "concentration_risk": 0.18,
    "liquidity_ratio": 0.85,
    "timestamp": "2025-07-16T10:30:00Z"
  }
}
```

3.1.4.2 Update Risk Limits

Endpoint: `PUT /risk/limits`

Description: Update risk management limits

Request Body:

```
{
  "max_var": 1000000.00,
  "max_leverage": 10.0,
  "max_concentration": 0.25,
  "max_drawdown": 0.20
}
```

Response:

```
{
  "status": "success",
  "data": {
    "message": "Risk limits updated successfully",
    "timestamp": "2025-07-16T10:30:00Z"
  }
}
```

3.1.5 Analytics API

3.1.5.1 Get Performance Metrics

Endpoint: `GET /analytics/performance`

Description: Retrieve performance analytics

Parameters: - `period` (optional): Time period (1d, 7d, 30d, 90d, 1y) - `metric` (optional): Specific metric (pnl, sharpe, volatility)

Response:

```json
{
  "status": "success",
  "data": {
    "period": "30d",
    "total_pnl": 15000.50,
    "sharpe_ratio": 2.15,
    "volatility": 0.18,
    "max_drawdown": 0.08,
    "win_rate": 0.67,
    "avg_trade_duration": 245.5,
    "timestamp": "2025-07-16T10:30:00Z"
  }
}
```

3.1.5.2 Get Arbitrage Opportunities

Endpoint: `GET /analytics/opportunities`

Description: Retrieve current arbitrage opportunities

Parameters: - `min_profit` (optional): Minimum profit threshold - `status` (optional): Filter by status (active, executed, expired)

Response:

```json
{
  "status": "success",
  "data": {
    "opportunities": [
      {
        "id": "opp_345678",
        "symbol": "BTC/USD",
        "type": "spatial",
        "buy_exchange": "binance",
        "sell_exchange": "okx",
        "buy_price": 44950.00,
        "sell_price": 45100.00,
        "profit": 150.00,
        "profit_percentage": 0.33,
        "confidence": 0.95,
        "timestamp": "2025-07-16T10:30:00Z",
        "status": "active"
      }
    ]
  }
}
```

3.2 WebSocket API

3.2.1 Connection

Endpoint: `wss://api.arbitrage-engine.com/v1/ws`

Authentication: JWT token via query parameter or header

Connection String: `wss://api.arbitrage-engine.com/v1/ws?token=<jwt_token>`

3.2.2 Message Format

Client to Server:

```
{
  "type": "subscribe",
  "channel": "prices",
  "symbol": "BTC/USD",
  "id": "req_123"
}
```

Server to Client:

```
{
  "type": "price_update",
  "channel": "prices",
  "symbol": "BTC/USD",
  "data": {
    "price": 45000.50,
    "volume": 1234.56,
    "timestamp": "2025-07-16T10:30:00Z"
  },
  "id": "req_123"
}
```

3.2.3 Subscription Channels

3.2.3.1 Price Updates **Channel**: `prices`

Description: Real-time price updates for instruments

Subscription:

```json
{
  "type": "subscribe",
  "channel": "prices",
  "symbol": "BTC/USD"
}
```

3.2.3.2 Order Updates **Channel**: `orders`

Description: Real-time order status updates

Subscription:

```json
{
  "type": "subscribe",
  "channel": "orders"
}
```

3.2.3.3 Risk Updates **Channel**: `risk`

Description: Real-time risk metric updates

Subscription:

```json
{
  "type": "subscribe",
  "channel": "risk"
}
```

3.2.3.4 Arbitrage Opportunities **Channel**: `opportunities`

Description: Real-time arbitrage opportunity notifications

Subscription:

```json
{
  "type": "subscribe",
  "channel": "opportunities",
  "min_profit": 100.00
}
```

4 Setup and Installation

4.1 System Requirements

4.1.1 Hardware Requirements

Minimum Requirements: - **CPU**: 8 cores (Intel i7-9700K / AMD Ryzen 7 3700X) - **RAM**: 16GB DDR4 - **Storage**: 500GB NVMe SSD - **Network**: 1Gbps ethernet connection

Recommended Requirements: - **CPU**: 16 cores (Intel i9-12900K / AMD Ryzen 9 5950X) - **RAM**: 32GB DDR4-3200 - **Storage**: 1TB NVMe SSD (Gen4) - **Network**: 10Gbps ethernet connection

4.1.2 Software Requirements

Operating System: - **Linux**: Ubuntu 20.04 LTS or CentOS 8+ - **Docker**: 20.10+ (for containerized deployment) - **Kubernetes**: 1.24+ (for orchestrated deployment)

Development Tools: - **Compiler**: GCC 11+ or Clang 14+ - **Build System**: CMake 3.20+ - **Version Control**: Git 2.25+

4.2 Development Environment Setup

4.2.1 Prerequisites Installation

Ubuntu/Debian:

```bash
# Update package repository
sudo apt update && sudo apt upgrade -y

# Install development tools
sudo apt install -y \
    build-essential \
    cmake \
    git \
    curl \
    wget \
    pkg-config \
    libssl-dev \
    zlib1g-dev

# Install GCC 11
sudo apt install -y gcc-11 g++-11
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-11 100
sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-11 100
```

CentOS/RHEL:

```
# Install development tools
sudo yum groupinstall -y "Development Tools"
sudo yum install -y cmake git curl wget openssl-devel zlib-devel

# Install GCC 11 (requires additional repository)
sudo yum install -y centos-release-scl
sudo yum install -y devtoolset-11-gcc devtoolset-11-gcc-c++
scl enable devtoolset-11 bash
```

4.2.2 Database Setup

PostgreSQL Installation:

```
# Ubuntu/Debian
sudo apt install -y postgresql postgresql-contrib

# Start and enable PostgreSQL
sudo systemctl start postgresql
sudo systemctl enable postgresql

# Create database and user
sudo -u postgres psql -c "CREATE DATABASE arbitrage_db;"
sudo -u postgres psql -c "CREATE USER arbitrage_user WITH PASSWORD 'secure_passw
sudo -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE arbitrage_db TO arbit
```

TimescaleDB Extension:

```
# Add TimescaleDB repository
echo "deb https://packagecloud.io/timescale/timescaledb/ubuntu/ $(lsb_release -c
wget --quiet -O - https://packagecloud.io/timescale/timescaledb/gpgkey | sudo ap
sudo apt update

# Install TimescaleDB
sudo apt install -y timescaledb-2-postgresql-14

# Enable extension in database
sudo -u postgres psql -d arbitrage_db -c "CREATE EXTENSION IF NOT EXISTS timesca
```

Redis Installation:

```
# Ubuntu/Debian
sudo apt install -y redis-server

# Start and enable Redis
sudo systemctl start redis-server
sudo systemctl enable redis-server
```

```
# Configure Redis for production
sudo sed -i 's/^# maxmemory <bytes>/maxmemory 4gb/' /etc/redis/redis.conf
sudo sed -i 's/^# maxmemory-policy noeviction/maxmemory-policy allkeys-lru/' /et
sudo systemctl restart redis-server
```

4.2.3 Message Queue Setup

Apache Kafka Installation:

```
# Download and install Kafka
wget https://downloads.apache.org/kafka/2.8.0/kafka_2.13-2.8.0.tgz
tar -xzf kafka_2.13-2.8.0.tgz
sudo mv kafka_2.13-2.8.0 /opt/kafka

# Create systemd service files
sudo tee /etc/systemd/system/zookeeper.service > /dev/null <<EOF
[Unit]
Description=Apache Zookeeper server
Documentation=http://zookeeper.apache.org
Requires=network.target remote-fs.target
After=network.target remote-fs.target

[Service]
Type=forking
User=kafka
Group=kafka
Environment=JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
ExecStart=/opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/kafka/config/zoo
ExecStop=/opt/kafka/bin/zookeeper-server-stop.sh
Restart=on-abnormal

[Install]
WantedBy=multi-user.target
EOF

sudo tee /etc/systemd/system/kafka.service > /dev/null <<EOF
[Unit]
Description=Apache Kafka Server
Documentation=http://kafka.apache.org/documentation.html
Requires=zookeeper.service

[Service]
Type=forking
```

```
User=kafka
Group=kafka
Environment=JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
ExecStart=/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.
ExecStop=/opt/kafka/bin/kafka-server-stop.sh
Restart=on-abnormal

[Install]
WantedBy=multi-user.target
EOF

# Start services
sudo systemctl daemon-reload
sudo systemctl enable zookeeper kafka
sudo systemctl start zookeeper kafka
```

4.2.4 Dependency Management

Install Boost Libraries:

```
# Ubuntu/Debian
sudo apt install -y libboost-all-dev

# Or build from source for latest version
wget https://boostorg.jfrog.io/artifactory/main/release/1.81.0/source/boost_1_81
tar -xzf boost_1_81_0.tar.gz
cd boost_1_81_0
./bootstrap.sh --prefix=/usr/local
sudo ./b2 install
```

Install Additional Dependencies:

```
# WebSocket++
git clone https://github.com/zaphoyd/websocketpp.git
cd websocketpp
mkdir build && cd build
cmake ..
make -j$(nproc)
sudo make install

# nlohmann/json
git clone https://github.com/nlohmann/json.git
cd json
mkdir build && cd build
cmake ..
```

```
make -j$(nproc)
sudo make install

# spdlog
git clone https://github.com/gabime/spdlog.git
cd spdlog
mkdir build && cd build
cmake ..
make -j$(nproc)
sudo make install
```

4.2.5 Environment Configuration

Create Configuration Directory:

```
sudo mkdir -p /etc/arbitrage-engine
sudo mkdir -p /var/log/arbitrage-engine
sudo mkdir -p /var/lib/arbitrage-engine
```

Configuration File:

```
sudo tee /etc/arbitrage-engine/config.json > /dev/null <<EOF
{
  "system": {
    "log_level": "INFO",
    "log_file": "/var/log/arbitrage-engine/system.log",
    "max_threads": 16,
    "memory_pool_size": "2GB"
  },
  "database": {
    "host": "localhost",
    "port": 5432,
    "database": "arbitrage_db",
    "username": "arbitrage_user",
    "password": "secure_password",
    "max_connections": 20
  },
  "redis": {
    "host": "localhost",
    "port": 6379,
    "password": "",
    "database": 0
  },
  "kafka": {
    "brokers": ["localhost:9092"],
    "topics": {
      "market_data": "market-data",
      "orders": "orders",
      "risk": "risk-events"
    }
  },
  "exchanges": {
    "binance": {
      "enabled": true,
      "api_key": "your_binance_api_key",
      "secret_key": "your_binance_secret_key",
      "websocket_url": "wss://stream.binance.com:9443/ws"
    },
    "okx": {
      "enabled": true,
      "api_key": "your_okx_api_key",
      "secret_key": "your_okx_secret_key",
      "websocket_url": "wss://ws.okx.com:8443/ws/v5/public"
    }
  },
  "risk_management": {
    "max_var": 1000000.0,
    "max_leverage": 10.0,
    "max_concentration": 0.25,
    "max_drawdown": 0.15
  }
}
EOF
```

4.3 Build Process

4.3.1 Source Code Compilation

Clone Repository:

```
git clone https://github.com/yourusername/Synthetic-Arbitrage-Detection-Engine.g
cd Synthetic-Arbitrage-Detection-Engine
```

Build with CMake:

```
# Create build directory
mkdir build && cd build

# Configure build
cmake -DCMAKE_BUILD_TYPE=Release \
      -DCMAKE_CXX_STANDARD=20 \
      -DBUILD_TESTS=ON \
      -DBUILD_BENCHMARKS=ON \
      ..

# Compile
make -j$(nproc)

# Run tests
ctest --verbose

# Install
sudo make install
```

Build Script:

```
#!/bin/bash
# build.sh

set -e

echo "Building Synthetic Arbitrage Detection Engine..."

# Clean previous build
rm -rf build
mkdir build
cd build

# Configure
cmake -DCMAKE_BUILD_TYPE=Release \
      -DCMAKE_CXX_STANDARD=20 \
      -DBUILD_TESTS=ON \
      -DBUILD_BENCHMARKS=ON \
      -DENABLE_SIMD=ON \
      -DENABLE_NUMA=ON \
      ..

# Build
make -j$(nproc)

# Test
echo "Running tests..."
ctest --verbose
```

```
# Benchmark
echo "Running benchmarks..."
./benchmarks/performance_benchmark

echo "Build completed successfully!"
```

4.3.2 Frontend Build

Node.js Setup:

```
# Install Node.js 18+
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt-get install -y nodejs

# Install Yarn
npm install -g yarn
```

Build Dashboard:

```
cd dashboard
yarn install
yarn build
```

5 Deployment Guide

5.1 Production Deployment

5.1.1 Docker Containerization

Dockerfile:

```
# Multi-stage build
FROM ubuntu:20.04 AS builder

# Install build dependencies
RUN apt-get update && apt-get install -y \
    build-essential \
    cmake \
    git \
    libboost-all-dev \
    libssl-dev \
    zlib1g-dev

# Copy source code
COPY . /app
WORKDIR /app

# Build application
RUN mkdir build && cd build && \
    cmake -DCMAKE_BUILD_TYPE=Release .. && \
    make -j$(nproc)

# Production image
FROM ubuntu:20.04

# Install runtime dependencies
RUN apt-get update && apt-get install -y \
    libboost-system1.71.0 \
    libboost-thread1.71.0 \
    libboost-filesystem1.71.0 \
    libssl1.1 \
    && rm -rf /var/lib/apt/lists/*

# Copy binary and configuration
COPY --from=builder /app/build/bin/arbitrage-engine /usr/local/bin/
COPY --from=builder /app/config/ /etc/arbitrage-engine/

# Create user
```

```
RUN groupadd -r arbitrage && useradd -r -g arbitrage arbitrage

# Set permissions
RUN chown -R arbitrage:arbitrage /etc/arbitrage-engine

# Expose port
EXPOSE 8080

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
  CMD curl -f http://localhost:8080/health || exit 1

# Run as non-root user
USER arbitrage

# Start application
CMD ["/usr/local/bin/arbitrage-engine", "--config", "/etc/arbitrage-engine/confi
```

Docker Compose:

```
version: '3.8'

services:
  arbitrage-engine:
    build: .
    ports:
      - "8080:8080"
    environment:
      - CONFIG_FILE=/etc/arbitrage-engine/config.json
    depends_on:
      - postgres
      - redis
      - kafka
    volumes:
      - ./config:/etc/arbitrage-engine
      - ./logs:/var/log/arbitrage-engine
    restart: unless-stopped

  postgres:
    image: timescale/timescaledb:latest-pg14
    environment:
      - POSTGRES_DB=arbitrage_db
      - POSTGRES_USER=arbitrage_user
      - POSTGRES_PASSWORD=secure_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data
    command: redis-server --appendonly yes

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

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

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./ssl:/etc/nginx/ssl
    depends_on:
      - arbitrage-engine
```

```
volumes:
  postgres_data:
  redis_data:
```

5.1.2 Kubernetes Deployment

Namespace:

```
apiVersion: v1
kind: Namespace
metadata:
  name: arbitrage-engine
```

ConfigMap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: arbitrage-config
  namespace: arbitrage-engine
data:
  config.json: |
    {
      "system": {
        "log_level": "INFO",
        "max_threads": 16,
        "memory_pool_size": "2GB"
      },
      "database": {
        "host": "postgres-service",
        "port": 5432,
        "database": "arbitrage_db",
        "username": "arbitrage_user",
        "password": "secure_password"
      }
    }
```

Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: arbitrage-engine
  namespace: arbitrage-engine
spec:
  replicas: 3
  selector:
    matchLabels:
      app: arbitrage-engine
  template:
    metadata:
      labels:
        app: arbitrage-engine
    spec:
      containers:
      - name: arbitrage-engine
        image: arbitrage-engine:latest
        ports:
        - containerPort: 8080
        env:
        - name: CONFIG_FILE
          value: /etc/arbitrage-engine/config.json
        volumeMounts:
        - name: config
          mountPath: /etc/arbitrage-engine
        resources:
          requests:
            memory: "4Gi"
            cpu: "2"
          limits:
            memory: "8Gi"
            cpu: "4"
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /ready
            port: 8080
          initialDelaySeconds: 5
          periodSeconds: 5
      volumes:
      - name: config
        configMap:
          name: arbitrage-config
```

Service:

```
apiVersion: v1
kind: Service
metadata:
  name: arbitrage-engine-service
  namespace: arbitrage-engine
spec:
  selector:
    app: arbitrage-engine
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
  type: ClusterIP
```

Ingress:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: arbitrage-engine-ingress
  namespace: arbitrage-engine
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  tls:
  - hosts:
    - api.arbitrage-engine.com
    secretName: arbitrage-tls
  rules:
  - host: api.arbitrage-engine.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: arbitrage-engine-service
            port:
              number: 80
```

5.1.3 Database Migration

Migration Script:

```sql
-- migrations/001_initial_schema.sql

-- Create hypertables for time-series data
CREATE TABLE IF NOT EXISTS market_data (
    id BIGSERIAL,
    symbol VARCHAR(20) NOT NULL,
    exchange VARCHAR(20) NOT NULL,
    price DECIMAL(20, 8) NOT NULL,
    volume DECIMAL(20, 8) NOT NULL,
    timestamp TIMESTAMPTZ NOT NULL,
    PRIMARY KEY (id, timestamp)
);

SELECT create_hypertable('market_data', 'timestamp');

-- Create index for efficient queries
CREATE INDEX IF NOT EXISTS idx_market_data_symbol_timestamp
ON market_data (symbol, timestamp DESC);

-- Create positions table
CREATE TABLE IF NOT EXISTS positions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    symbol VARCHAR(20) NOT NULL,
    side VARCHAR(10) NOT NULL,
    size DECIMAL(20, 8) NOT NULL,
    entry_price DECIMAL(20, 8) NOT NULL,
    current_price DECIMAL(20, 8),
    unrealized_pnl DECIMAL(20, 8),
    realized_pnl DECIMAL(20, 8),
    status VARCHAR(20) NOT NULL,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Create orders table
CREATE TABLE IF NOT EXISTS orders (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    symbol VARCHAR(20) NOT NULL,
    side VARCHAR(10) NOT NULL,
    type VARCHAR(20) NOT NULL,
    quantity DECIMAL(20, 8) NOT NULL,
    price DECIMAL(20, 8),
    filled_quantity DECIMAL(20, 8) DEFAULT 0,
    average_price DECIMAL(20, 8),
    status VARCHAR(20) NOT NULL,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Create risk_metrics table
CREATE TABLE IF NOT EXISTS risk_metrics (
    id BIGSERIAL,
    portfolio_var DECIMAL(20, 8),
    expected_shortfall DECIMAL(20, 8),
    max_drawdown DECIMAL(10, 6),
    leverage_ratio DECIMAL(10, 4),
    concentration_risk DECIMAL(10, 6),
    timestamp TIMESTAMPTZ NOT NULL,
    PRIMARY KEY (id, timestamp)
);

SELECT create_hypertable('risk_metrics', 'timestamp');
```

Migration Runner:

```
#!/bin/bash
# migrate.sh

set -e

DB_HOST=${DB_HOST:-localhost}
DB_PORT=${DB_PORT:-5432}
DB_NAME=${DB_NAME:-arbitrage_db}
DB_USER=${DB_USER:-arbitrage_user}
DB_PASSWORD=${DB_PASSWORD:-secure_password}

export PGPASSWORD=$DB_PASSWORD

echo "Running database migrations..."

for migration in migrations/*.sql; do
    echo "Applying migration: $migration"
    psql -h $DB_HOST -p $DB_PORT -U $DB_USER -d $DB_NAME -f "$migration"
```

```
done

echo "Database migration completed successfully!"
```

5.1.4 SSL/TLS Configuration

Generate SSL Certificate:

```
# Self-signed certificate for development
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
    -keyout arbitrage-engine.key \
    -out arbitrage-engine.crt \
    -subj "/C=US/ST=State/L=City/O=Organization/CN=api.arbitrage-engine.com"

# Create Kubernetes secret
kubectl create secret tls arbitrage-tls \
    --cert=arbitrage-engine.crt \
    --key=arbitrage-engine.key \
    -n arbitrage-engine
```

NGINX Configuration:

```
upstream arbitrage_backend {
    server arbitrage-engine:8080;
}

server {
    listen 80;
    server_name api.arbitrage-engine.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name api.arbitrage-engine.com;

    ssl_certificate /etc/nginx/ssl/arbitrage-engine.crt;
    ssl_certificate_key /etc/nginx/ssl/arbitrage-engine.key;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384;
    ssl_prefer_server_ciphers off;

    location / {
        proxy_pass http://arbitrage_backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /ws {
        proxy_pass http://arbitrage_backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

6 Monitoring and Maintenance

6.1 System Monitoring

6.1.1 Prometheus Configuration

prometheus.yml:

```yaml
global:
  scrape_interval: 15s
  evaluation_interval: 15s

rule_files:
  - "arbitrage_rules.yml"

scrape_configs:
  - job_name: 'arbitrage-engine'
    static_configs:
      - targets: ['localhost:8080']
    metrics_path: '/metrics'
    scrape_interval: 5s

  - job_name: 'postgres'
    static_configs:
      - targets: ['localhost:9187']

  - job_name: 'redis'
    static_configs:
      - targets: ['localhost:9121']

  - job_name: 'kafka'
    static_configs:
      - targets: ['localhost:9308']

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

Alert Rules:

```yaml
# arbitrage_rules.yml
groups:
- name: arbitrage_alerts
  rules:
```

```yaml
  - alert: HighLatency
    expr: arbitrage_engine_latency_p99 > 0.01
    for: 2m
    labels:
      severity: warning
    annotations:
      summary: "High latency detected"
      description: "99th percentile latency is {{ $value }}s"

  - alert: SystemDown
    expr: up{job="arbitrage-engine"} == 0
    for: 1m
    labels:
      severity: critical
    annotations:
      summary: "Arbitrage engine is down"
      description: "Arbitrage engine has been down for more than 1 minute"

  - alert: HighMemoryUsage
    expr: arbitrage_engine_memory_usage_percent > 80
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: "High memory usage"
      description: "Memory usage is {{ $value }}%"

  - alert: RiskLimitBreach
    expr: arbitrage_engine_var_utilization > 0.9
    for: 1m
    labels:
      severity: critical
    annotations:
      summary: "Risk limit breach"
      description: "VaR utilization is {{ $value }}%"
```

6.1.2 Grafana Dashboard

Dashboard Configuration:

```json
{
  "dashboard": {
    "title": "Arbitrage Engine Dashboard",
    "panels": [
      {
        "title": "System Performance",
        "type": "graph",
        "targets": [
          {
            "expr": "arbitrage_engine_latency_p99",
            "legendFormat": "99th Percentile Latency"
          },
          {
            "expr": "arbitrage_engine_throughput",
            "legendFormat": "Throughput (ops/sec)"
          }
        ]
      },
      {
        "title": "Risk Metrics",
        "type": "singlestat",
        "targets": [
          {
            "expr": "arbitrage_engine_var_current",
            "legendFormat": "Current VaR"
          }
        ]
      },
      {
        "title": "Active Positions",
        "type": "table",
        "targets": [
          {
            "expr": "arbitrage_engine_positions_count",
            "legendFormat": "Position Count"
          }
        ]
      }
    ]
  }
}
```

6.2 Log Management

6.2.1 Centralized Logging

Logstash Configuration:

```
input {
  beats {
    port => 5044
  }
}

filter {
  if [fileset][module] == "arbitrage" {
    grok {
      match => { "message" => "%{TIMESTAMP_ISO8601:timestamp} \[%{LOGLEVEL:level
    }

    date {
      match => [ "timestamp", "ISO8601" ]
    }

    if [level] == "ERROR" {
      mutate {
        add_tag => [ "error" ]
      }
    }
  }
}

output {
  elasticsearch {
    hosts => ["elasticsearch:9200"]
    index => "arbitrage-engine-%{+YYYY.MM.dd}"
  }
}
```

Filebeat Configuration:

```
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /var/log/arbitrage-engine/*.log
  fields:
    service: arbitrage-engine
  multiline.pattern: '^\d{4}-\d{2}-\d{2}'
  multiline.negate: true
  multiline.match: after

output.logstash:
  hosts: ["logstash:5044"]

processors:
- add_host_metadata: ~
- add_fields:
    target: ""
    fields:
      environment: production
```

6.3 Database Maintenance

6.3.1 Backup Strategy

Automated Backup Script:

```
#!/bin/bash
# backup.sh

set -e

BACKUP_DIR="/var/backups/arbitrage-engine"
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
DB_NAME="arbitrage_db"
DB_USER="arbitrage_user"
RETENTION_DAYS=30

mkdir -p $BACKUP_DIR

# Create database backup
echo "Creating database backup..."
pg_dump -h localhost -U $DB_USER -d $DB_NAME -f "$BACKUP_DIR/database_$TIMESTAMP

# Compress backup
gzip "$BACKUP_DIR/database_$TIMESTAMP.sql"

# Create application backup
echo "Creating application backup..."
tar -czf "$BACKUP_DIR/application_$TIMESTAMP.tar.gz" /etc/arbitrage-engine
```

```
# Clean old backups
find $BACKUP_DIR -name "*.gz" -mtime +$RETENTION_DAYS -delete

echo "Backup completed successfully!"
```

Restore Script:

```
#!/bin/bash
# restore.sh

set -e

BACKUP_FILE=$1
DB_NAME="arbitrage_db"
DB_USER="arbitrage_user"

if [ -z "$BACKUP_FILE" ]; then
    echo "Usage: $0 <backup_file.sql.gz>"
    exit 1
fi

echo "Restoring database from $BACKUP_FILE..."

# Extract backup
gunzip -c "$BACKUP_FILE" | psql -h localhost -U $DB_USER -d $DB_NAME

echo "Database restore completed successfully!"
```

6.3.2 Performance Optimization

Database Maintenance:

```
-- Analyze table statistics
ANALYZE market_data;
ANALYZE positions;
ANALYZE orders;

-- Reindex tables
REINDEX TABLE market_data;
REINDEX TABLE positions;
REINDEX TABLE orders;

-- Update table statistics
UPDATE pg_stat_user_tables SET n_tup_upd = 0, n_tup_del = 0;
```

```
-- Vacuum tables
VACUUM ANALYZE market_data;
VACUUM ANALYZE positions;
VACUUM ANALYZE orders;
```

TimescaleDB Maintenance:

```
-- Set retention policy for market_data
SELECT add_retention_policy('market_data', INTERVAL '30 days');

-- Compress old data
SELECT add_compression_policy('market_data', INTERVAL '7 days');

-- Show chunk information
SELECT chunk_name, range_start, range_end, is_compressed
FROM timescaledb_information.chunks
WHERE hypertable_name = 'market_data'
ORDER BY range_start DESC;
```

6.4 Security Hardening

6.4.1 System Security

Firewall Configuration:

```
# Enable UFW
sudo ufw enable

# Allow SSH
sudo ufw allow 22/tcp

# Allow HTTP/HTTPS
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp

# Allow application port
sudo ufw allow 8080/tcp

# Allow database access (internal only)
sudo ufw allow from 10.0.0.0/8 to any port 5432

# Deny all other traffic
sudo ufw default deny incoming
sudo ufw default allow outgoing
```

System Hardening:

```
# Disable root login
sudo sed -i 's/PermitRootLogin yes/PermitRootLogin no/' /etc/ssh/sshd_config

# Enable key-based authentication
sudo sed -i 's/#PubkeyAuthentication yes/PubkeyAuthentication yes/' /etc/ssh/ssh
sudo sed -i 's/#PasswordAuthentication yes/PasswordAuthentication no/' /etc/ssh/

# Restart SSH service
sudo systemctl restart ssh

# Install fail2ban
sudo apt install -y fail2ban
sudo systemctl enable fail2ban
sudo systemctl start fail2ban
```

6.4.2 Application Security

JWT Configuration:

```
{
  "jwt": {
    "secret": "your-256-bit-secret-key-here",
    "expiration": "24h",
    "issuer": "arbitrage-engine",
    "audience": "arbitrage-api"
  },
  "rate_limiting": {
    "requests_per_minute": 1000,
    "burst_limit": 100
  },
  "cors": {
    "allowed_origins": ["https://dashboard.arbitrage-engine.com"],
    "allowed_methods": ["GET", "POST", "PUT", "DELETE"],
    "allowed_headers": ["Authorization", "Content-Type"]
  }
}
```

6.5 Troubleshooting Guide

6.5.1 Common Issues

High Latency:

```
# Check system resources
top
```

```bash
htop
iostat -x 1

# Check network latency
ping -c 10 exchange.com
traceroute exchange.com

# Check database connections
psql -h localhost -U arbitrage_user -d arbitrage_db -c "SELECT * FROM pg_stat_ac

# Check application logs
tail -f /var/log/arbitrage-engine/system.log
```

Memory Issues:

```bash
# Check memory usage
free -m
cat /proc/meminfo

# Check for memory leaks
valgrind --tool=memcheck --leak-check=full ./arbitrage-engine

# Monitor memory usage
watch -n 1 'ps aux --sort=-%mem | head -10'
```

Database Performance:

```sql
-- Check slow queries
SELECT query, mean_time, calls, total_time
FROM pg_stat_statements
ORDER BY mean_time DESC
LIMIT 10;

-- Check index usage
SELECT schemaname, tablename, attname, n_distinct, correlation
FROM pg_stats
WHERE tablename = 'market_data';

-- Check connection status
SELECT state, count(*)
FROM pg_stat_activity
GROUP BY state;
```

6.5.2 Performance Tuning

System Optimization:

```
# Increase file descriptor limits
echo "arbitrage soft nofile 65536" >> /etc/security/limits.conf
echo "arbitrage hard nofile 65536" >> /etc/security/limits.conf

# Optimize network settings
echo "net.core.rmem_default = 262144" >> /etc/sysctl.conf
echo "net.core.wmem_default = 262144" >> /etc/sysctl.conf
echo "net.core.rmem_max = 16777216" >> /etc/sysctl.conf
echo "net.core.wmem_max = 16777216" >> /etc/sysctl.conf
sysctl -p

# Set CPU affinity
taskset -c 0-3 ./arbitrage-engine
```

Database Tuning:

```
-- PostgreSQL configuration
ALTER SYSTEM SET shared_buffers = '2GB';
ALTER SYSTEM SET effective_cache_size = '8GB';
ALTER SYSTEM SET work_mem = '64MB';
ALTER SYSTEM SET maintenance_work_mem = '512MB';
ALTER SYSTEM SET checkpoint_segments = 32;
ALTER SYSTEM SET wal_buffers = '16MB';
SELECT pg_reload_conf();
```

7 Appendices

7.1 Appendix A: Configuration Reference

7.1.1 Complete Configuration Schema

```json
{
  "system": {
    "log_level": "INFO|DEBUG|WARN|ERROR",
    "log_file": "/var/log/arbitrage-engine/system.log",
    "max_threads": 16,
    "memory_pool_size": "2GB",
    "enable_simd": true,
    "enable_numa": true,
    "cpu_affinity": [0, 1, 2, 3]
  },
  "database": {
    "host": "localhost",
    "port": 5432,
    "database": "arbitrage_db",
    "username": "arbitrage_user",
    "password": "secure_password",
    "ssl_mode": "require",
    "max_connections": 20,
    "connection_timeout": 30,
    "query_timeout": 60
  },
  "redis": {
    "host": "localhost",
    "port": 6379,
    "password": "",
    "database": 0,
    "timeout": 5,
    "retry_attempts": 3
  },
  "kafka": {
    "brokers": ["localhost:9092"],
    "topics": {
      "market_data": "market-data",
      "orders": "orders",
      "risk": "risk-events",
      "analytics": "analytics"
    },
    "consumer_group": "arbitrage-engine",
    "batch_size": 1000,
    "timeout": 10
  },
  "api": {
    "host": "0.0.0.0",
    "port": 8080,
    "ssl_enabled": true,
    "ssl_cert": "/etc/ssl/arbitrage-engine.crt",
    "ssl_key": "/etc/ssl/arbitrage-engine.key",
    "cors_enabled": true,
    "rate_limit": {
      "requests_per_minute": 1000,
      "burst_limit": 100
    }
  },
  "exchanges": {
    "binance": {
      "enabled": true,
      "api_key": "your_binance_api_key",
      "secret_key": "your_binance_secret_key",
      "websocket_url": "wss://stream.binance.com:9443/ws",
      "rest_url": "https://api.binance.com",
      "rate_limit": 1200,
      "timeout": 5
    },
    "okx": {
      "enabled": true,
      "api_key": "your_okx_api_key",
      "secret_key": "your_okx_secret_key",
      "websocket_url": "wss://ws.okx.com:8443/ws/v5/public",
      "rest_url": "https://www.okx.com/api/v5",
      "rate_limit": 600,
      "timeout": 5
    },
    "bybit": {
      "enabled": false,
      "api_key": "your_bybit_api_key",
      "secret_key": "your_bybit_secret_key",
      "websocket_url": "wss://stream.bybit.com/v5/public/spot",
      "rest_url": "https://api.bybit.com/v5",
      "rate_limit": 600,
      "timeout": 5
    }
  },
  "risk_management": {
    "max_var": 1000000.0,
    "max_leverage": 10.0,
    "max_concentration": 0.25,
    "max_drawdown": 0.15,
    "var_confidence_level": 0.95,
    "risk_check_interval": 1,
    "stop_loss_enabled": true,
    "stop_loss_percentage": 0.02
  },
  "monitoring": {
    "metrics_enabled": true,
    "metrics_port": 9090,
    "health_check_interval": 30,
    "alert_email": "admin@arbitrage-engine.com",
    "alert_webhook": "https://hooks.slack.com/services/..."
  }
}
```

7.2 Appendix B: API Error Codes

7.2.1 HTTP Status Codes

Code	Description	Example
200	OK	Successful request
201	Created	Resource created successfully
400	Bad Request	Invalid request parameters
401	Unauthorized	Invalid authentication token
403	Forbidden	Insufficient permissions
404	Not Found	Resource not found
409	Conflict	Resource already exists
429	Too Many Requests	Rate limit exceeded
500	Internal Server Error	Server error
503	Service Unavailable	Service temporarily unavailable

7.2.2 Application Error Codes

Code	Category	Description
1001	Authentication	Invalid API key
1002	Authentication	Expired token

Code	Category	Description
1003	Authentication	Insufficient permissions
2001	Market Data	Symbol not found
2002	Market Data	Exchange not available
2003	Market Data	Data not available
3001	Trading	Invalid order parameters
3002	Trading	Insufficient balance
3003	Trading	Order not found
3004	Trading	Order already filled
4001	Risk	Risk limit exceeded
4002	Risk	Position size too large
4003	Risk	Leverage too high
5001	System	Database connection error
5002	System	External service unavailable
5003	System	Configuration error

7.3 Appendix C: Performance Benchmarks

7.3.1 System Performance Metrics

Latency Benchmarks:

```
Operation: Price Calculation
- Mean: 0.05ms
- P50: 0.04ms
- P95: 0.08ms
- P99: 0.12ms
- P99.9: 0.25ms

Operation: Order Execution
- Mean: 2.1ms
- P50: 1.8ms
- P95: 3.5ms
- P99: 5.2ms
- P99.9: 8.7ms

Operation: Risk Calculation
- Mean: 0.8ms
- P50: 0.6ms
- P95: 1.5ms
- P99: 2.3ms
- P99.9: 4.1ms
```

Throughput Benchmarks:

```
Market Data Processing: 50,000 updates/sec
Order Processing: 10,000 orders/sec
Risk Calculations: 25,000 calculations/sec
Database Operations: 15,000 queries/sec
```

7.4 Appendix D: Security Checklist

7.4.1 Pre-Deployment Security

- ☐ All default passwords changed
- ☐ SSL/TLS certificates installed
- ☐ Firewall rules configured
- ☐ Database access restricted
- ☐ API rate limiting enabled
- ☐ JWT tokens configured
- ☐ Audit logging enabled
- ☐ Regular security updates scheduled

7.4.2 Runtime Security

- ☐ Monitor failed login attempts
- ☐ Check for unusual API usage
- ☐ Verify SSL certificate expiration
- ☐ Monitor system resource usage
- ☐ Review access logs regularly
- ☐ Backup encryption verified
- ☐ Incident response plan tested

Document Classification: Internal Use Only **Version**: 1.0 **Last Updated**: July 16, 2025 **Next Review**: January 16, 2026
