End-to-End System Design for Stock Price Prediction

1. Data Collection & Ingestion

Technology Choices

- **Market Data APIs**:
  - **Primary**: Alpha Vantage and Yahoo Finance APIs for historical and real-time market data
  - **Secondary**: Quandl for extended financial datasets
- **Alternative Data Sources**:
  - News API for sentiment analysis (NewsAPI.org)
  - Social media sentiment analysis via Twitter API
  - SEC EDGAR filings via SEC API
- **Ingestion Infrastructure**:
  - Apache Airflow for orchestrating data collection workflows
  - Apache Kafka for real-time data streaming

Justification

Alpha Vantage and Yahoo Finance offer reliable, low-latency access to market data with generous free tiers suitable for initial development. Kafka provides a scalable streaming solution that decouples data producers from consumers, allowing multiple downstream systems to process the same data without interference.

Tradeoffs

- **Reliability vs Cost**: Premium financial data services (Bloomberg, Reuters) offer more reliable data but at significantly higher costs
- **Latency vs Completeness**: Real-time streaming APIs may have occasional data gaps compared to end-of-day batch data
- **Build vs Buy**: Building connectors to multiple sources increases development time but provides greater flexibility than using a single vendor solution

2. Data Processing Pipeline

Technology Choices

- **Data Cleaning & Validation**:
    - Apache Spark for distributed data processing
    - Great Expectations for data quality validation
- **Feature Engineering**:
    - Spark ML for distributed feature computation
    - Feature Store (Feast) for feature management and serving
- **Data Storage**:
    - Raw data: AWS S3 (data lake)
    - Processed data: Snowflake (data warehouse)
    - Feature vectors: Redis (for low-latency feature serving)

Justification

Spark provides scalable data processing capabilities suitable for our time-series financial data. A feature store centralizes feature definitions, enables feature reuse, and separates feature computation from model training. Snowflake offers scalable storage with SQL interface for analytics while Redis provides low-latency feature vector access for real-time prediction.

Tradeoffs

- **Batch vs Streaming**: Initial focus on batch processing simplifies development but adds latency
- **Computation cost vs Response time**: Pre-computing features increases storage costs but improves prediction latency
- **Specialized vs General-purpose tools**: Specialized time-series processing libraries might offer better performance for specific operations but reduce flexibility

3. Model Operations

Technology Choices

- **Model Training**:
    - MLflow for experiment tracking and model versioning
    - Kubernetes for distributed training jobs
    - GPU instances for accelerated training

- **Model Deployment**:
    - Docker containers for model packaging
    - KubeFlow for model orchestration
    - NVIDIA Triton for model serving
- **Model Monitoring**:
    - Prometheus for metric collection
    - Grafana for visualization
    - Evidently AI for ML-specific monitoring

Justification

MLflow provides comprehensive experiment tracking and model registry capabilities with minimal overhead. Containerizing models ensures consistent environment across development and production. Prometheus and Grafana are industry standards for monitoring with large community support.

Tradeoffs

- **Training frequency vs resource usage**: Daily retraining consumes more resources but captures market shifts better than weekly retraining
- **Model complexity vs inference speed**: Complex ensemble models may improve accuracy but increase prediction latency
- **Automated vs human-in-the-loop deployment**: Fully automated deployment accelerates updates but increases risk without human validation

4. Insight Delivery

Technology Choices

- **Web Dashboard**:
    - React.js for frontend UI
    - D3.js for interactive visualizations
    - Material UI for component library
- **API Layer**:
    - FastAPI for REST API development
    - API Gateway for rate limiting and authentication
- **Notification System**:
    - AWS SNS for alert notifications
    - Scheduled reports via email using AWS SES

Justification

React provides a responsive UI framework with large component ecosystem. FastAPI enables rapid API development with automatic documentation. Material UI ensures consistent styling with minimal development effort.

Tradeoffs

- **Rich UI vs performance**: More interactive dashboards increase engagement but can slow rendering on mobile devices
- **Push vs pull updates**: Real-time updates (WebSockets) improve freshness but increase server load
- **General vs personalized insights**: Targeted insights require more complex user profiling but deliver higher value

5. System Considerations

Scalability

- **Vertical scaling** for database performance and model training
- **Horizontal scaling** for API serving and data processing
- **Auto-scaling** groups based on load patterns
- **Serverless** components for sporadic workloads

Reliability

- **Multi-AZ deployment** for high availability
- **Circuit breakers** for external API dependencies
- **Graceful degradation** strategies for component failures
- **Data replication** across regions for disaster recovery

Latency Requirements

- **End-to-end pipeline latency**: <15 minutes from market data to prediction
- **API response time**: <200ms for prediction endpoints
- **Dashboard loading**: <2 seconds for initial load, <500ms for updates

Cost Considerations

- **Reserved instances** for predictable workloads
- **Spot instances** for fault-tolerant batch processing
- **Caching strategies** to reduce redundant computations
- **Data lifecycle policies** to archive/delete old data

6. Data Flow

Batch Processing Flow

1. Daily market data collection job runs after market close
2. Raw data written to S3 data lake
3. Spark jobs clean data and compute features
4. Features written to feature store
5. Model retraining triggered weekly or on performance degradation
6. Batch predictions generated for all tracked stocks
7. Insights derived and stored for dashboard consumption

Real-time Flow

1. Market data streams continuously during trading hours
2. Stream processing updates technical indicators in near real-time
3. On-demand prediction API retrieves latest features and generates predictions
4. Alerts triggered when prediction crosses configured thresholds
5. Dashboard updates via polling or WebSocket push

7. Implementation Challenges & Mitigations

Challenge 1: Data Quality Issues

- **Potential Impact**: Poor data leads to inaccurate predictions
- **Mitigation**:
  - Implement comprehensive data validation with Great Expectations
  - Cross-validate across multiple data sources
  - Alert on anomalous data patterns
  - Implement reconciliation with trusted reference data

Challenge 2: Model Drift

- **Potential Impact**: Degrading prediction accuracy over time
- **Mitigation**:
  - Monitor statistical properties of input features
  - Track prediction error metrics over time
  - Implement automated retraining with performance thresholds
  - Maintain shadow deployment of candidate models

Challenge 3: System Scalability During Market Volatility

- **Potential Impact**: System failure during critical high-volume periods
- **Mitigation**:
  - Load testing with 10x normal volume
  - Circuit breakers and throttling mechanisms
  - Priority queues for critical operations
  - Over-provisioned infrastructure during market hours

Challenge 4: Compliance and Security

- **Potential Impact**: Regulatory issues or data breaches
- **Mitigation**:
  - Encrypt sensitive data at rest and in transit
  - Implement comprehensive audit logging
  - Role-based access control for all system components
  - Regular security reviews and penetration testing

Challenge 5: Cost Management

- **Potential Impact**: Excessive operational costs
- **Mitigation**:
  - Resource utilization monitoring
  - Automated scaling based on usage patterns
  - Cost allocation tagging and budgeting
  - Performance optimization for compute-intensive components