Arsitektur Sistem Platform Komoditas Watch Indonesia

Tanggal: 7 Juni 2025
Versi: 1.0
Disusun oleh: Tim Arsitektur Sistem

📝 Ringkasan Eksekutif

Platform Komoditas Watch Indonesia adalah sistem monitoring komoditas pangan berbasis web dan mobile yang dirancang untuk mengintegrasikan multiple data sources, menyediakan analisis real-time, dan menawarkan prediksi berbasis machine learning untuk pengendalian inflasi komoditas. Arsitektur yang diusulkan mengadopsi pendekatan microservices dengan implementasi domain-driven design, memaksimalkan skalabilitas, maintainability, dan security.

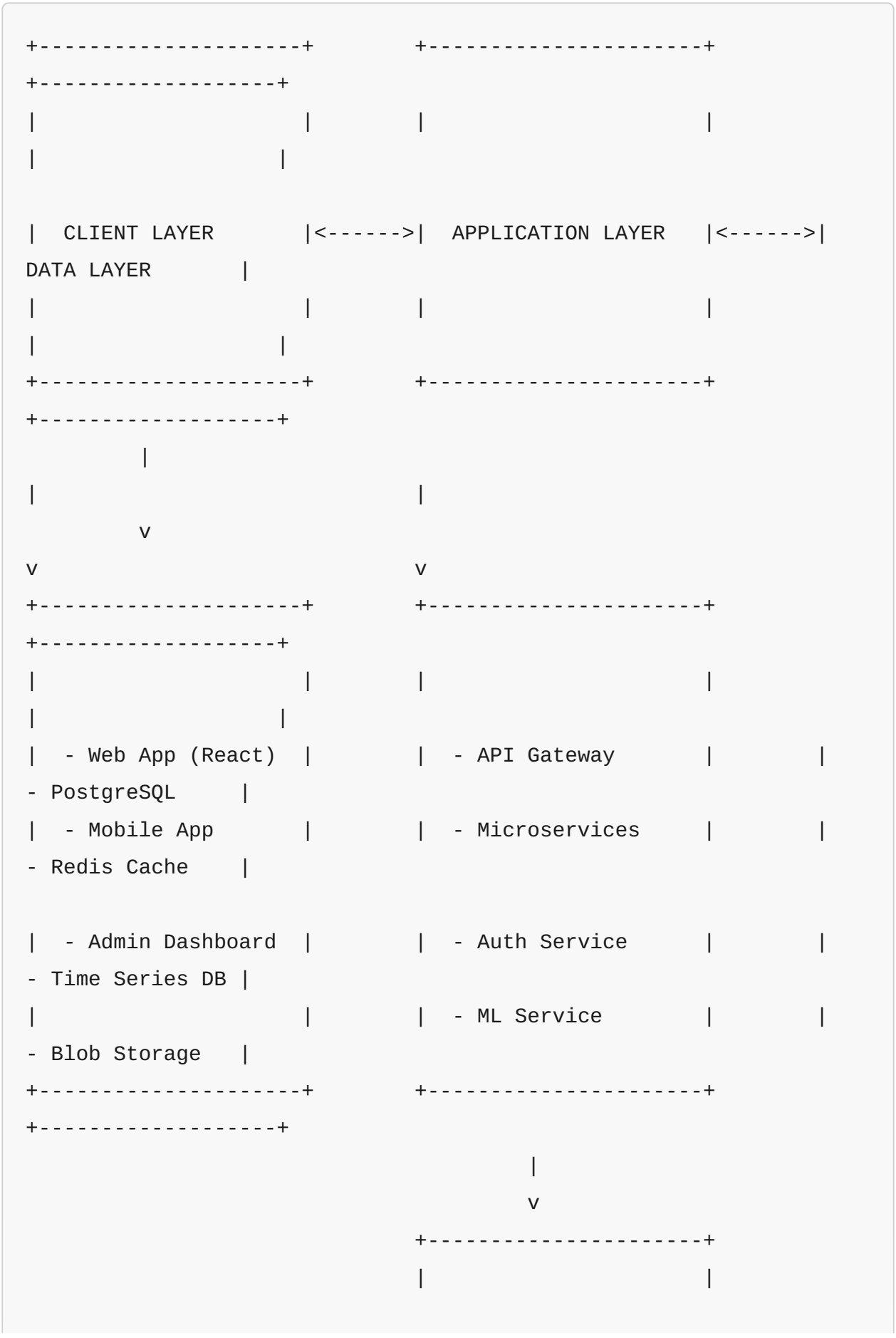
Sistem ini dirancang untuk mengintegrasikan 3+ data sources utama (BPS, BMKG, dan Global Commodities) dengan kapasitas analisis canggih untuk 8+ komoditas strategis. Arsitektur ini mendukung fitur real-time dashboard, early warning system, predictive analytics, dan role-based access untuk beragam stakeholder.

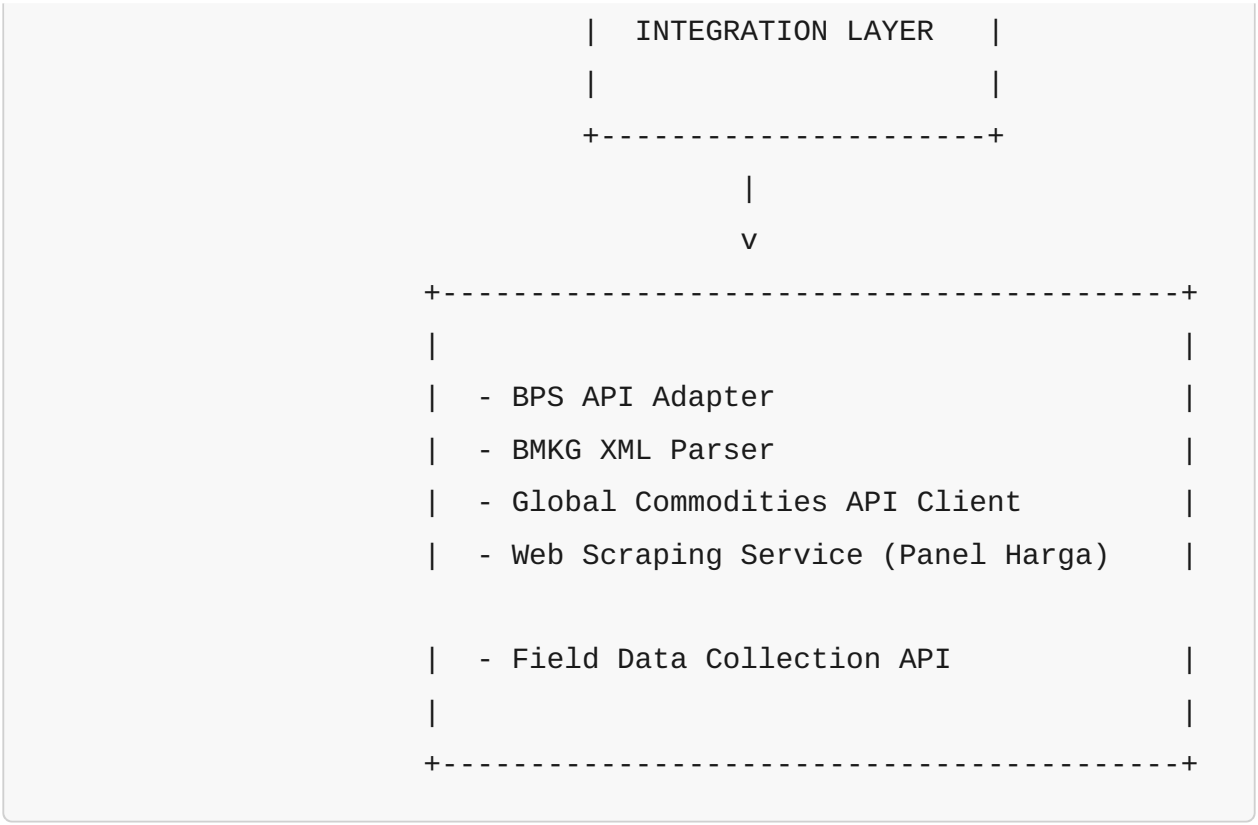
🔍 Daftar Isi

🏗 Arsitektur Sistem Tingkat Tinggi

High-Level Architecture

Komponen Utama


CLIENT LAYER
APPLICATION LAYER
DATA LAYER
- Web App (React)
- Mobile App
- Admin Dashboard
- API Gateway
- Microservices
- Auth Service
- ML Service
- PostgreSQL
- Redis Cache
- Time Series DB
- Blob Storage



Arsitektur Logika

Platform ini menggunakan desain arsitektur berbasis microservices dengan domain-driven design. Komponen utama diorganisir secara loosely coupled namun memiliki sentralisasi business logic pada domain services.

1. Lapisan Klien

- **Web Application**: React.js dengan TailwindCSS untuk UI responsif
- **Mobile Application**: React Native dengan shared component library
- **Admin Dashboard**: React dengan template admin khusus

2. Lapisan Aplikasi

- **API Gateway**: Entrypoint terpusat untuk semua requests
- **Microservices**: Backend services berdasarkan domain bisnis
- **Authentication Service**: Pengelolaan user, roles dan permissions
- **Machine Learning Service**: Prediksi harga dan deteksi anomali

3. Lapisan Integrasi

- **Data Integration Services**: Adapter untuk external APIs
- **ETL Pipeline**: Extraksi, transformasi dan loading data
- **Messaging System**: Asynchronous communication backbone

4. Lapisan Data

- **Relational Database**: PostgreSQL untuk structured data
- **Time Series Database**: Untuk data historis harga dan metrics
- **Cache Layer**: Redis untuk performa dan real-time data
- **Blob Storage**: Untuk penyimpanan file dan reports

5. Cross-Cutting Concerns

- **Monitoring**: System health, metrics, dan logging
- **Security**: Access control, encryption, dan auditing
- **DevOps**: CI/CD pipeline, containerization, dan orchestration

---

🧩 Arsitektur Komponen

Frontend Architecture

Web Application (React.js)

```
/src
|-- /assets            # Static assets (images, fonts, etc.)

|-- /components        # Shared UI components
|   |-- /common        # Common UI elements (buttons, inputs, etc.)
|   |-- /charts        # Chart components (Recharts/ECharts)

|   |-- /maps          # Map components (react-leaflet)
|   |-- /layout        # Layout components (header, footer,
sidebar)
|   |-- /forms         # Form components and validation

|   `-- /tables        # Table components
|
|-- /hooks             # Custom React hooks
|-- /contexts          # React context providers

|-- /features          # Feature-based modules
|   |-- /dashboard     # Dashboard feature
|   |-- /commodities   # Commodity details feature

|   |-- /predictions   # ML predictions feature
|   |-- /alerts        # Alerts and notifications feature
|   |-- /reports       # Report generation feature

|   `-- /admin         # Admin feature
|
|-- /services          # API services
|   |-- /api           # API clients

|   |-- /auth          # Authentication service
|   |-- /socket        # WebSocket service
|   `-- /storage       # Local storage service
|
|-- /utils             # Utility functions
```

```
|-- /constants          # Constants and enumerations
|-- /types              # TypeScript interfaces and types
|-- /styles             # Global styles and Tailwind configuration
|-- /locales            # Internationalization files
|-- /routes             # Routing configuration
|-- App.tsx             # Main application component
`-- index.tsx           # Entry point
```

Teknologi dan Library

- **Framework**: React.js 18.x dengan TypeScript
- **State Management**: React Context API + React Query
- **Styling**: TailwindCSS dengan custom theme
- **Data Visualization**:
- **Charts**: Recharts (responsive) dan ECharts (complex visualizations)
- **Maps**: react-leaflet dengan custom GeoJSON layers
- **Form Management**: React Hook Form dengan Yup validation
- **Internationalization**: i18next
- **Authentication**: JWT dengan secure HTTP-only cookies
- **HTTP Client**: Axios dengan interceptors
- **Real-time Updates**: Socket.IO client
- **Testing**: Jest, React Testing Library
- **Build Tools**: Vite

Backend Architecture

Microservices Architecture

```
backend/

|-- /gateway                # API Gateway
|   |-- server.js           # Main application file
|   |-- /routes             # Routes configuration
|   |-- /middleware         # Middleware functions

|   `-- /config             # Configuration files
|
|-- /services               # Domain-specific microservices
|   |-- /user-service       # User management service

|   |-- /commodity-service  # Commodity data service
|   |-- /weather-service    # Weather data service
|   |-- /analytics-service  # Analytics service

|   |-- /alert-service      # Alert and notification service
|   |-- /report-service     # Report generation service
|   `-- /admin-service      # Admin management service
|

|-- /integration            # Integration services
|   |-- /bps-service        # BPS API integration
|   |-- /bmkg-service       # BMKG API integration
|   |-- /commodities-service # Global Commodities API integration

|   `-- /scraper-service    # Web scraping service
|
|-- /ml-service             # Machine Learning service
|   |-- /api                # FastAPI application

|   |-- /models             # ML models and pipelines
|   |-- /data               # Data processing utilities
|   `-- /config             # ML service configuration
|
```

```
|-- /shared                  # Shared utilities and modules
|   |-- /utils               # Utility functions
|   |-- /models              # Shared data models
|   |-- /middleware          # Shared middleware

|   `-- /constants           # Shared constants
|
|-- /libs                    # Internal libraries
|   |-- /message-bus         # Message bus implementation

|   |-- /logger              # Logging library
|   |-- /validator           # Data validation library
|   `-- /error-handler       # Error handling library
|
`-- /infrastructure          # Infrastructure components
    |-- /database            # Database migrations and seeds
    |-- /cache               # Cache configuration
    |-- /messaging           # Message queue configuration
    |-- /monitoring          # Monitoring configuration
    `-- /security            # Security configuration
```

Service Breakdown

1. **API Gateway**
   - Routing dan service discovery
   - Authentication dan authorization
   - Rate limiting dan request throttling
   - Request validation
   - API documentation (Swagger/OpenAPI)

2. **User Service**
   - User registration dan management
   - Authentication (JWT)
   - Profile management
   - Role-based access control
   - User preferences

3. **Commodity Service**
   - Commodity data management
   - Price tracking
   - Historical data
   - Commodity relationships dan taxonomy
   - Supply chain tracking

4. **Weather Service**
   - Weather data processing
   - Weather forecasts
   - Weather-to-commodity impact analysis
   - Geographical mapping of weather conditions

5. **Analytics Service**
   - Data aggregation dan analysis
   - Trend identification
   - Statistical calculations
   - Chart and visualization data

6. **Alert Service**
   - Price anomaly detection
   - Notification management
   - Alert rules dan thresholds
   - Delivery channels (email, SMS, push)

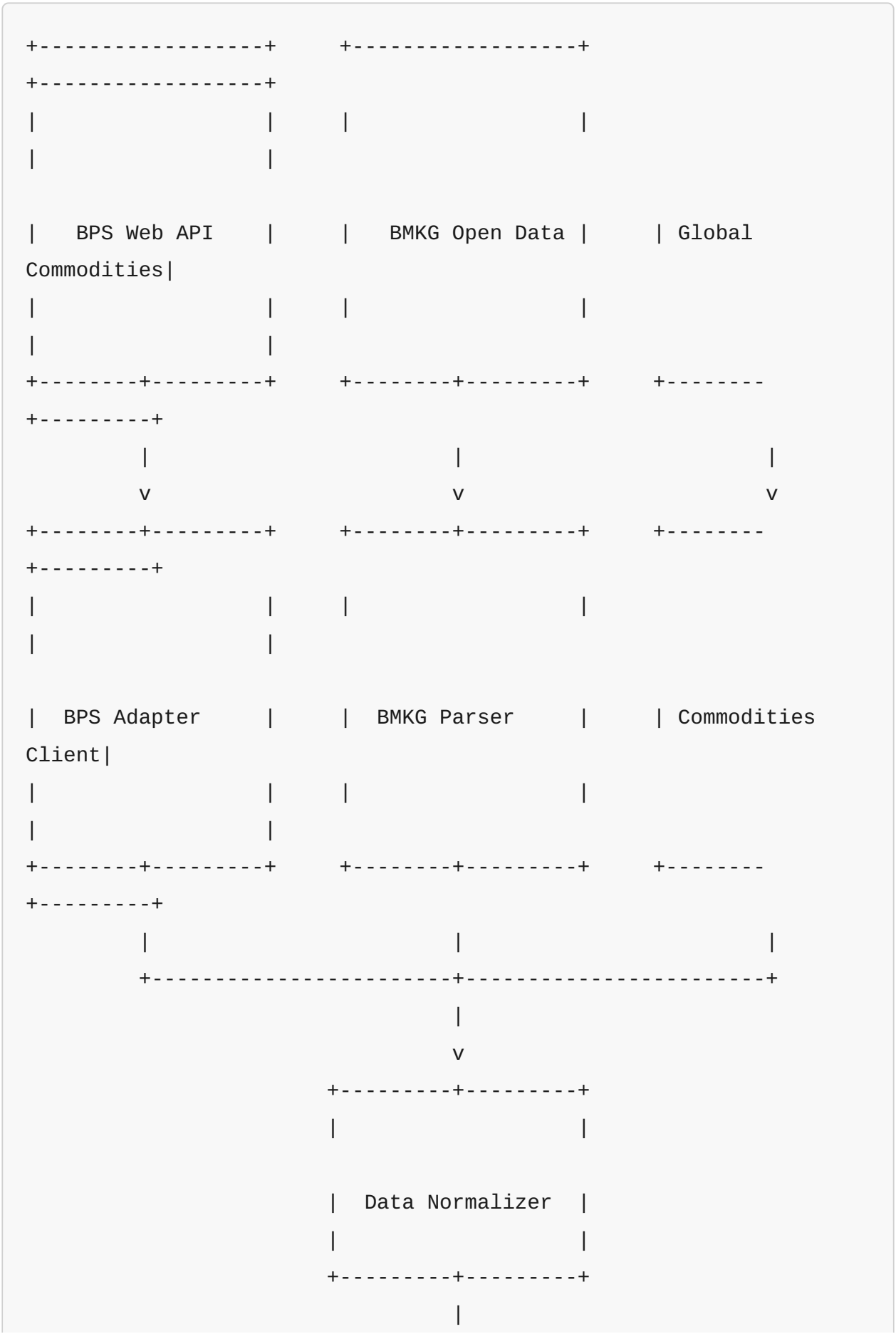
7. **Report Service**
   - Report template management
   - Dynamic report generation
   - Scheduled reports
   - Export formats (PDF, Excel)

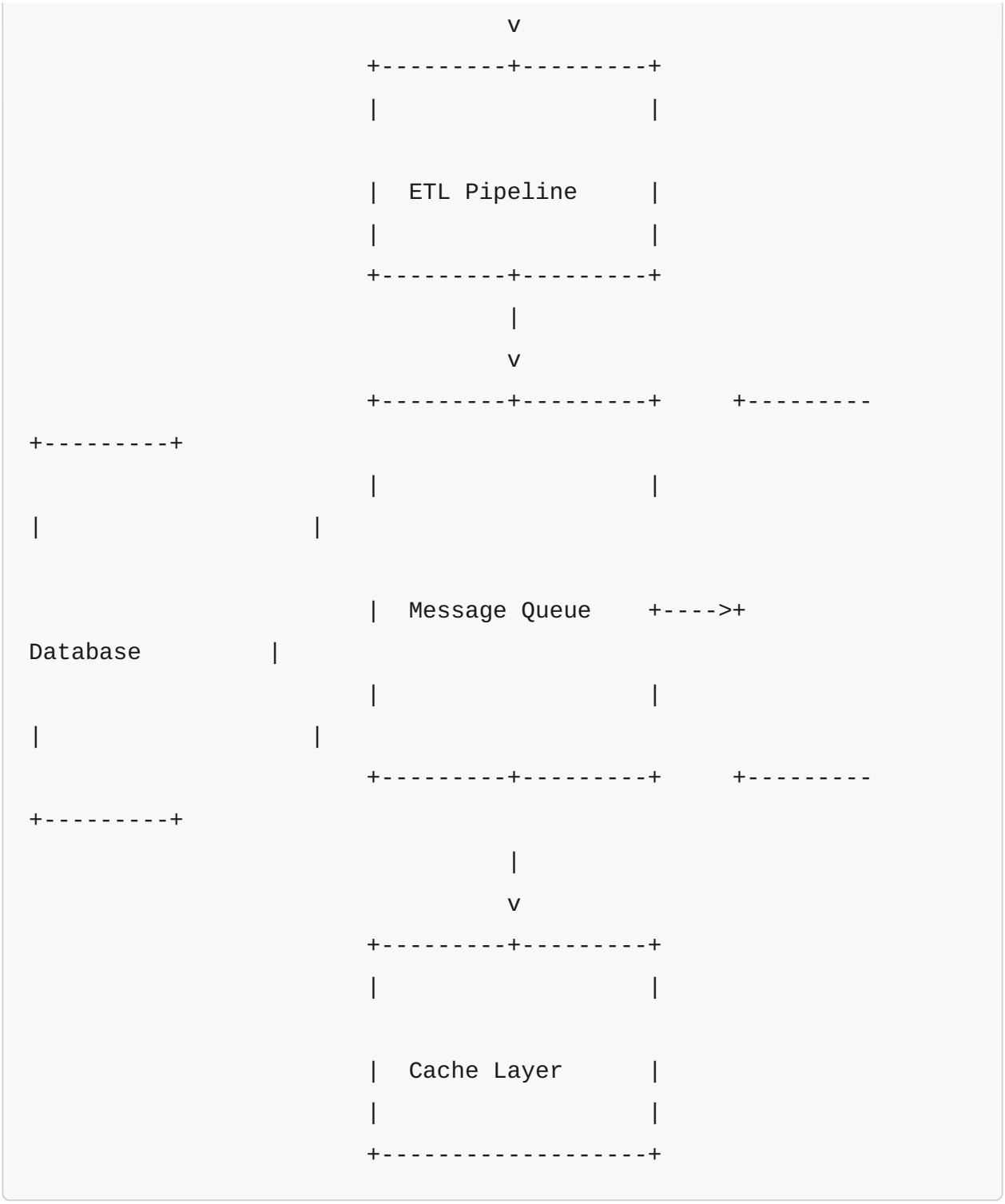
8. **Admin Service**
   - Platform configuration
   - User management
   - System monitoring
   - Audit logging

Teknologi dan Framework

- **Runtime**: Node.js 20.x
- **Framework**: Express.js untuk REST API
- **ORM**: Prisma untuk database access
- **Validation**: Joi/Zod
- **Authentication**: Passport.js with JWT
- **Caching**: Redis
- **Messaging**: RabbitMQ
- **API Documentation**: Swagger/OpenAPI
- **Testing**: Jest, Supertest
- **Logging**: Winston, Morgan
- **Process Management**: PM2

Data Integration Layer


BPS Web API
BMKG Open Data
Global Commodities
BPS Adapter
BMKG Parser
Commodities Client
Data Normalizer


ETL Pipeline
Message Queue
Database
Cache Layer

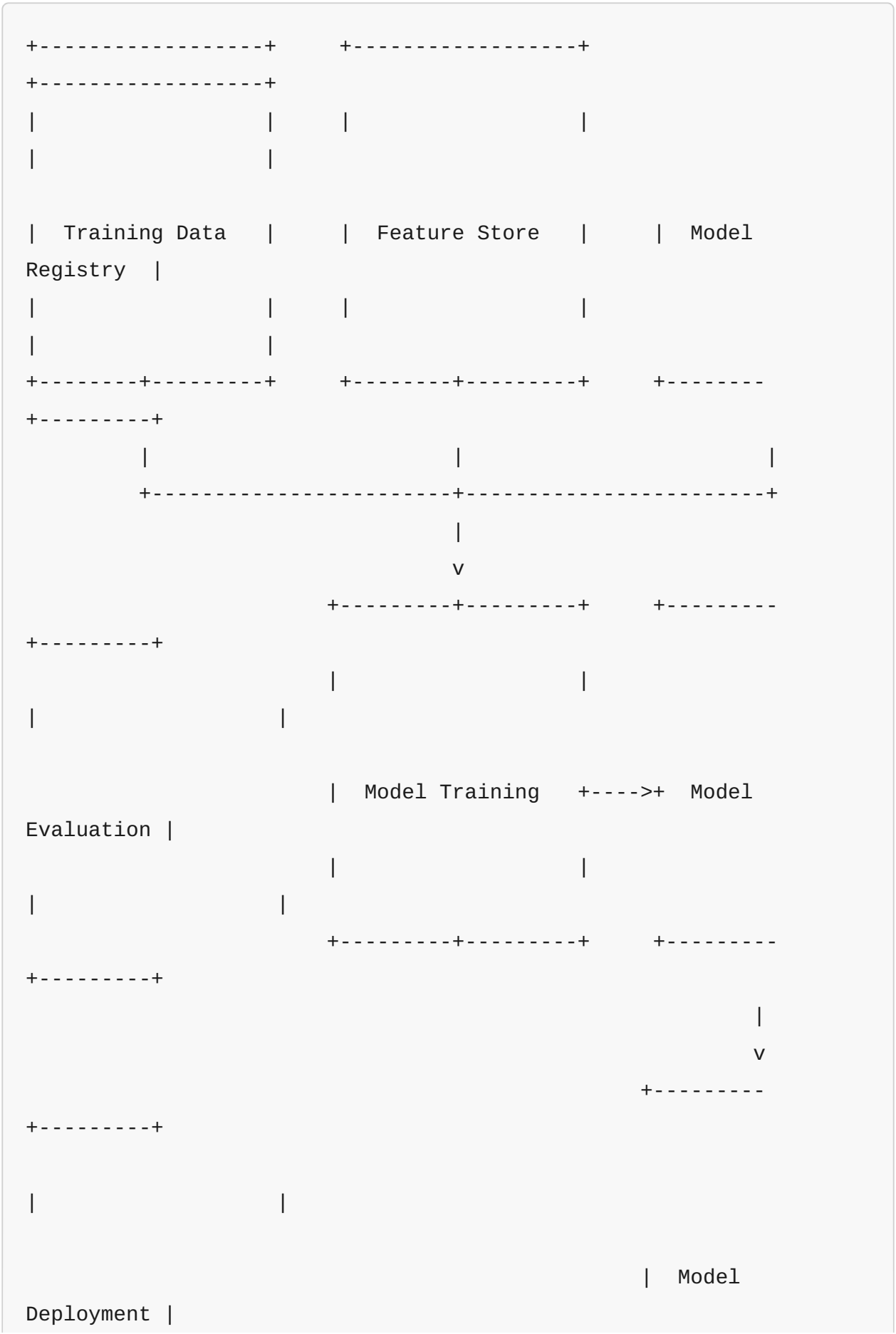
Komponen Integrasi

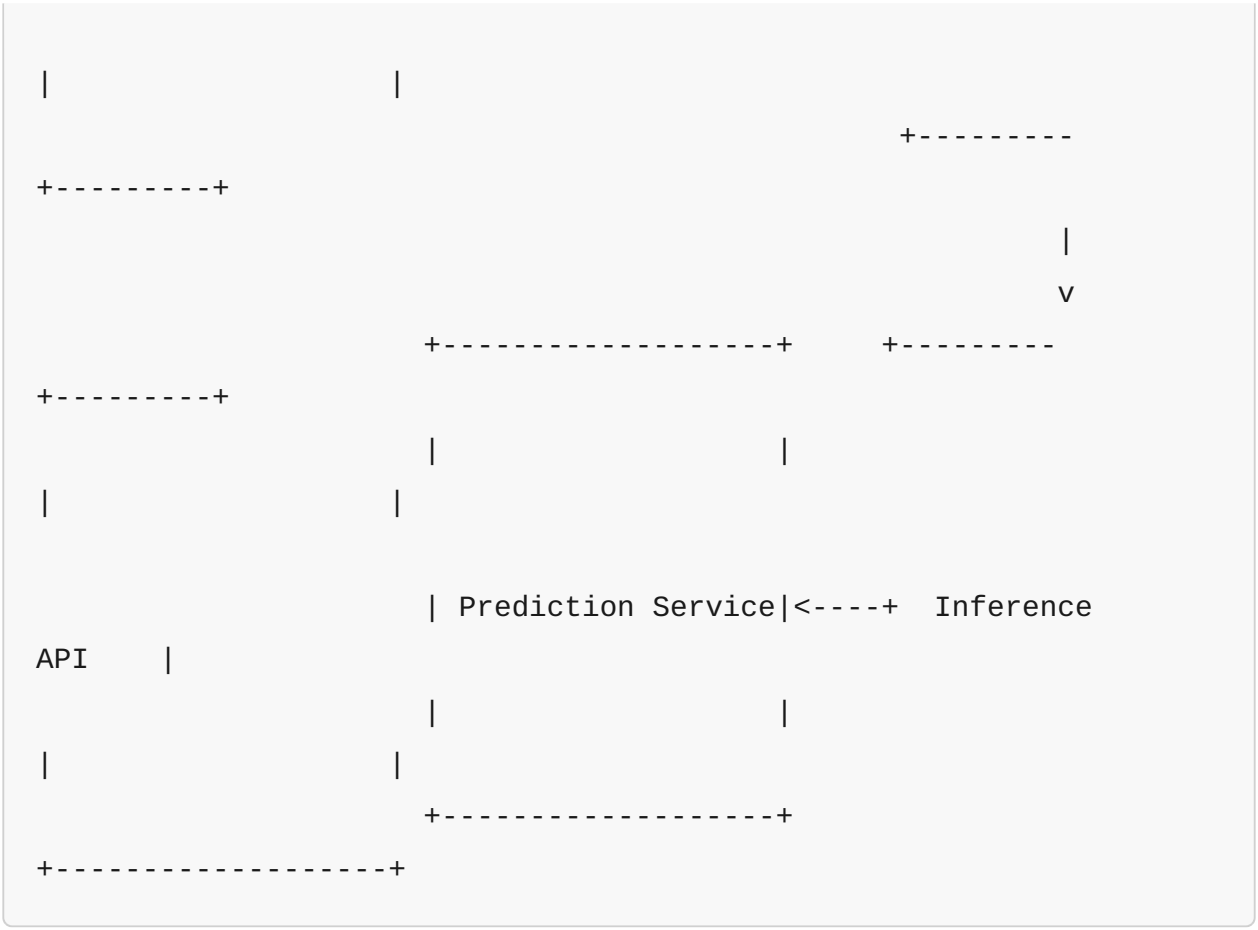
1. **BPS Adapter**
   - Mengintegrasikan BPS Web API
   - Menangani authentication dan rate limiting
   - Mendukung semua endpoint yang dibutuhkan:
     - Consumer Price Index (ID: 2212)
     - Wholesale Price Index (ID: 2498)
     - Trade data (Ekspor-Impor)
     - Implementasi caching dan error handling
2. **BMKG Parser**
   - Mengintegrasikan BMKG Open Data
   - Mem-parsing response format XML
   - Mendukung 34 provinsi
   - Mengekstrak data cuaca yang relevan untuk pertanian
3. **Global Commodities Client**
   - Mengintegrasikan Global Commodities API
   - Mendukung 13 komoditas dan 158 mata uang
   - Mendapatkan harga real-time dan historis
4. **Panel Harga Scraper**
   - Web scraping untuk Panel Harga Pangan
   - Extraction data terstruktur dari HTML
   - Scheduler untuk regular scraping
   - Proxy rotation untuk menghindari blocking
5. **Data Normalizer**
   - Menstandardisasi format data dari berbagai sumber
   - Mapping data fields ke skema standar
   - Validasi dan cleansing data
   - Menangani inconsistencies dan edge cases
6. **ETL Pipeline**
   - Proses ETL (Extract, Transform, Load)
   - Data enrichment dan augmentation
   - Historical data loading
   - Incremental updates

Strategi Implementasi

- **API Access Management**:
- Credential rotation dan secure storage
- Circuit breaker untuk external API
- Fallback mechanisms untuk service outages
- Adaptive retry policies
- **Data Processing Workflow**:
- Scheduled jobs untuk periodic data
- Stream processing untuk real-time data
- Batch processing untuk historical data
- Error recovery dan idempotent operations
- **Monitoring dan Alerting**:
- Data quality monitoring
- Integration health checks
- Alerting untuk integration failures
- Performance metrics

Machine Learning Pipeline


Training Data
Feature Store
Model Registry
Model Training
Model Evaluation
Model Deployment



ML Pipeline Components

1. **Data Preparation**
   - Feature engineering
   - Data cleaning dan preprocessing
   - Time series preprocessing
   - Train/validation/test splitting
2. **Training Pipeline**
   - Model training workflow
   - Hyperparameter optimization
   - Experiment tracking
   - Model versioning
3. **Model Registry**
   - Model storage dan versioning
   - Model metadata
   - A/B testing configuration
   - Deployment management

4. **Inference Service**
   - Real-time prediction endpoint
   - Batch prediction
   - Explanation API
   - Feature importance

ML Models dan Algorithms

1. **Price Forecasting**
   - **Primary Models**: Prophet (baseline), LSTM (deep learning)
   - **Features**:
     - Historical price data (daily, weekly, monthly)
     - Seasonal patterns
     - Weather data
     - Global commodity prices
     - Supply chain indicators
     - **Output**: Price forecasts for 7, 14, dan 30 days
2. **Anomaly Detection**
   - **Primary Algorithms**: DBSCAN, Isolation Forest
   - **Features**:
     - Price volatility
     - Supply chain disruptions
     - Weather anomalies
     - Market events
     - **Output**: Anomaly score, classification, dan explanation

3. **Distribution Optimization**
   - **Primary Algorithms**: Constraint Optimization, Mixed Integer Programming
   - **Features**:
     - Supply dan demand by region
     - Transportation costs
     - Storage capacity
     - Weather impact on logistics
     - **Output**: Optimal distribution paths dan quantities

ML Service Architecture

- **Framework**: FastAPI untuk ML API
- **Model Serving**: TensorFlow Serving / PyTorch Serve
- **Experiment Tracking**: MLflow
- **Feature Store**: Custom PostgreSQL implementation
- **Batch Processing**: Apache Airflow
- **GPU Support**: For LSTM training
- **Monitoring**: Prometheus + Grafana

💾 Database Schema

Relational Schema (PostgreSQL)

Core Entities

1. **Users**

```
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    full_name VARCHAR(255) NOT NULL,
    organization VARCHAR(255),
    role_id UUID NOT NULL REFERENCES roles(id),
    is_active BOOLEAN DEFAULT true,
    phone_number VARCHAR(50),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    last_login TIMESTAMP WITH TIME ZONE
);

CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_role_id ON users(role_id);
```

1. **Roles**

```
CREATE TABLE roles (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name VARCHAR(50) UNIQUE NOT NULL,
    description TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

INSERT INTO roles (name, description) VALUES
('admin', 'System administrator'),
('regulator', 'Government regulator'),
('distributor', 'Food distributor'),
('farmer', 'Agricultural producer');
```

1. **Permissions**

```
CREATE TABLE permissions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name VARCHAR(100) UNIQUE NOT NULL,
    description TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE TABLE role_permissions (
    role_id UUID NOT NULL REFERENCES roles(id) ON DELETE CASCADE,
    permission_id UUID NOT NULL REFERENCES permissions(id) ON
DELETE CASCADE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    PRIMARY KEY (role_id, permission_id)
);

CREATE INDEX idx_role_permissions_role_id ON
role_permissions(role_id);
```

1. **Commodities**

```
CREATE TABLE commodities (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    code VARCHAR(50) UNIQUE NOT NULL,
    name VARCHAR(100) NOT NULL,
    category_id UUID NOT NULL REFERENCES commodity_categories(id),
    description TEXT,
    unit VARCHAR(50) NOT NULL,
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_commodities_category_id ON
commodities(category_id);
```

1. **Commodity Categories**

```
CREATE TABLE commodity_categories (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name VARCHAR(100) NOT NULL,
    description TEXT,
    parent_id UUID REFERENCES commodity_categories(id),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_commodity_categories_parent_id ON
commodity_categories(parent_id);
```

1. **Regions**

```
CREATE TABLE regions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    code VARCHAR(20) UNIQUE NOT NULL,
    name VARCHAR(100) NOT NULL,
    type VARCHAR(50) NOT NULL, -- 'country', 'province',
'district', 'city'
    parent_id UUID REFERENCES regions(id),
    latitude DECIMAL(10, 8),
    longitude DECIMAL(11, 8),
    geometry GEOMETRY(POLYGON, 4326),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_regions_parent_id ON regions(parent_id);
CREATE INDEX idx_regions_geom ON regions USING GIST(geometry);
```

Data Entities

1. **Price Data**

```
CREATE TABLE price_data (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    commodity_id UUID NOT NULL REFERENCES commodities(id),
    region_id UUID NOT NULL REFERENCES regions(id),
    price DECIMAL(18, 2) NOT NULL,
    date DATE NOT NULL,
    source VARCHAR(50) NOT NULL, -- 'BPS', 'Panel Harga', 'Field
Data', etc.
    source_id VARCHAR(100), -- ID in the source system
    is_verified BOOLEAN DEFAULT false,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_price_data_commodity_id ON
price_data(commodity_id);
CREATE INDEX idx_price_data_region_id ON price_data(region_id);
CREATE INDEX idx_price_data_date ON price_data(date);
CREATE INDEX idx_price_data_source ON price_data(source);
```

1. **Weather Data**

```
CREATE TABLE weather_data (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    region_id UUID NOT NULL REFERENCES regions(id),
    date DATE NOT NULL,
    timestamp TIMESTAMP WITH TIME ZONE NOT NULL,
    temperature DECIMAL(5, 2),
    humidity DECIMAL(5, 2),
    rainfall DECIMAL(8, 2),
    wind_speed DECIMAL(5, 2),
    weather_condition VARCHAR(50),
    source VARCHAR(50) NOT NULL, -- 'BMKG', etc.
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_weather_data_region_id ON weather_data(region_id);
CREATE INDEX idx_weather_data_date ON weather_data(date);
```

1. **Production Data**

```
CREATE TABLE production_data (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    commodity_id UUID NOT NULL REFERENCES commodities(id),
    region_id UUID NOT NULL REFERENCES regions(id),
    year INT NOT NULL,
    month INT NOT NULL,
    production_volume DECIMAL(18, 2) NOT NULL,
    area_harvested DECIMAL(18, 2),
    productivity DECIMAL(10, 4),
    source VARCHAR(50) NOT NULL, -- 'BPS', 'Kementan', 'Field
Data', etc.
    is_verified BOOLEAN DEFAULT false,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_production_data_commodity_id ON
production_data(commodity_id);
CREATE INDEX idx_production_data_region_id ON
production_data(region_id);
CREATE INDEX idx_production_data_year_month ON
production_data(year, month);
```

1. **Supply Chain Data**

```
CREATE TABLE supply_chain_data (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    commodity_id UUID NOT NULL REFERENCES commodities(id),
    origin_region_id UUID NOT NULL REFERENCES regions(id),
    destination_region_id UUID NOT NULL REFERENCES regions(id),
    quantity DECIMAL(18, 2) NOT NULL,
    date DATE NOT NULL,
    transportation_mode VARCHAR(50),
    transportation_cost DECIMAL(18, 2),
    is_verified BOOLEAN DEFAULT false,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_supply_chain_data_commodity_id ON
supply_chain_data(commodity_id);
CREATE INDEX idx_supply_chain_data_origin_region_id ON
supply_chain_data(origin_region_id);
CREATE INDEX idx_supply_chain_data_destination_region_id ON
supply_chain_data(destination_region_id);
CREATE INDEX idx_supply_chain_data_date ON supply_chain_data(date);
```

ML and Prediction Entities

1. **Price Predictions**

```
CREATE TABLE price_predictions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    commodity_id UUID NOT NULL REFERENCES commodities(id),
    region_id UUID NOT NULL REFERENCES regions(id),
    prediction_date DATE NOT NULL,
    target_date DATE NOT NULL,
    predicted_price DECIMAL(18, 2) NOT NULL,
    confidence_lower DECIMAL(18, 2),
    confidence_upper DECIMAL(18, 2),
    model_version VARCHAR(50) NOT NULL,
    features JSONB,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_price_predictions_commodity_id ON
price_predictions(commodity_id);
CREATE INDEX idx_price_predictions_region_id ON
price_predictions(region_id);
CREATE INDEX idx_price_predictions_prediction_date ON
price_predictions(prediction_date);
CREATE INDEX idx_price_predictions_target_date ON
price_predictions(target_date);
```

1. **Anomaly Detection**

```
CREATE TABLE anomaly_detections (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    commodity_id UUID NOT NULL REFERENCES commodities(id),
    region_id UUID NOT NULL REFERENCES regions(id),
    date DATE NOT NULL,
    is_anomaly BOOLEAN NOT NULL,
    anomaly_score DECIMAL(10, 4) NOT NULL,
    anomaly_type VARCHAR(50), -- 'price_spike', 'supply_shortage',
etc.
    description TEXT,
    features JSONB,
    model_version VARCHAR(50) NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_anomaly_detections_commodity_id ON
anomaly_detections(commodity_id);
CREATE INDEX idx_anomaly_detections_region_id ON
anomaly_detections(region_id);
CREATE INDEX idx_anomaly_detections_date ON
anomaly_detections(date);
CREATE INDEX idx_anomaly_detections_is_anomaly ON
anomaly_detections(is_anomaly);
```

Alert and Notification Entities

1. **Alert Rules**

```
CREATE TABLE alert_rules (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name VARCHAR(100) NOT NULL,
    description TEXT,
    commodity_id UUID REFERENCES commodities(id),
    region_id UUID REFERENCES regions(id),
    rule_type VARCHAR(50) NOT NULL, -- 'price_threshold',
'anomaly', 'trend', etc.
    conditions JSONB NOT NULL,
    is_active BOOLEAN DEFAULT true,
    created_by UUID NOT NULL REFERENCES users(id),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_alert_rules_commodity_id ON
alert_rules(commodity_id);
CREATE INDEX idx_alert_rules_region_id ON alert_rules(region_id);
CREATE INDEX idx_alert_rules_rule_type ON alert_rules(rule_type);
```

1. **Alerts**

```
CREATE TABLE alerts (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    alert_rule_id UUID NOT NULL REFERENCES alert_rules(id),
    trigger_time TIMESTAMP WITH TIME ZONE NOT NULL,
    commodity_id UUID NOT NULL REFERENCES commodities(id),
    region_id UUID NOT NULL REFERENCES regions(id),
    alert_data JSONB NOT NULL,
    severity VARCHAR(20) NOT NULL, -- 'low', 'medium', 'high',
'critical'
    is_resolved BOOLEAN DEFAULT false,
    resolved_at TIMESTAMP WITH TIME ZONE,
    resolved_by UUID REFERENCES users(id),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_alerts_alert_rule_id ON alerts(alert_rule_id);
CREATE INDEX idx_alerts_commodity_id ON alerts(commodity_id);
CREATE INDEX idx_alerts_region_id ON alerts(region_id);
CREATE INDEX idx_alerts_severity ON alerts(severity);
CREATE INDEX idx_alerts_is_resolved ON alerts(is_resolved);
```

1. **Notifications**

```
CREATE TABLE notifications (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID NOT NULL REFERENCES users(id),
    alert_id UUID REFERENCES alerts(id),
    title VARCHAR(255) NOT NULL,
    message TEXT NOT NULL,
    notification_type VARCHAR(50) NOT NULL, -- 'alert', 'system',
'price_update', etc.
    is_read BOOLEAN DEFAULT false,
    read_at TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_notifications_user_id ON notifications(user_id);
CREATE INDEX idx_notifications_alert_id ON notifications(alert_id);
CREATE INDEX idx_notifications_is_read ON notifications(is_read);
```

Reporting and Auditing Entities

1. **Report Templates**

```
CREATE TABLE report_templates (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name VARCHAR(100) NOT NULL,
    description TEXT,
    template_data JSONB NOT NULL,
    is_active BOOLEAN DEFAULT true,
    created_by UUID NOT NULL REFERENCES users(id),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

1. **Generated Reports**

```
CREATE TABLE generated_reports (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    template_id UUID NOT NULL REFERENCES report_templates(id),
    report_name VARCHAR(255) NOT NULL,
    parameters JSONB,
    file_path VARCHAR(255) NOT NULL,
    file_type VARCHAR(20) NOT NULL, -- 'pdf', 'excel', etc.
    generated_by UUID NOT NULL REFERENCES users(id),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_generated_reports_template_id ON
generated_reports(template_id);
CREATE INDEX idx_generated_reports_generated_by ON
generated_reports(generated_by);
```

1. **Audit Logs**

```
CREATE TABLE audit_logs (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES users(id),
    action VARCHAR(50) NOT NULL, -- 'create', 'update', 'delete',
'login', etc.
    entity_type VARCHAR(50) NOT NULL, -- 'user', 'commodity',
'price_data', etc.
    entity_id UUID,
    old_values JSONB,
    new_values JSONB,
    ip_address VARCHAR(45),
    user_agent TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_audit_logs_user_id ON audit_logs(user_id);
CREATE INDEX idx_audit_logs_action ON audit_logs(action);
CREATE INDEX idx_audit_logs_entity_type ON audit_logs(entity_type);
CREATE INDEX idx_audit_logs_created_at ON audit_logs(created_at);
```

Time Series Database Schema

Untuk data deret waktu dengan volume tinggi (harga, cuaca), kami menggunakan TimescaleDB extension pada PostgreSQL:

```
-- Enable TimescaleDB extension
CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;

-- Create hypertable for high-frequency price data
CREATE TABLE price_timeseries (
    time TIMESTAMPTZ NOT NULL,
    commodity_id UUID NOT NULL,
    region_id UUID NOT NULL,
    price DECIMAL(18, 2) NOT NULL,
    source VARCHAR(50) NOT NULL,
    source_id VARCHAR(100),
    is_verified BOOLEAN DEFAULT false
);

-- Convert to hypertable partitioned by time
SELECT create_hypertable('price_timeseries', 'time');

-- Create indices
CREATE INDEX idx_price_timeseries_commodity_id ON
price_timeseries(commodity_id);
CREATE INDEX idx_price_timeseries_region_id ON
price_timeseries(region_id);
CREATE INDEX idx_price_timeseries_source ON
price_timeseries(source);

-- Create hypertable for weather data
CREATE TABLE weather_timeseries (
    time TIMESTAMPTZ NOT NULL,
    region_id UUID NOT NULL,
    temperature DECIMAL(5, 2),
    humidity DECIMAL(5, 2),
    rainfall DECIMAL(8, 2),
    wind_speed DECIMAL(5, 2),
    weather_condition VARCHAR(50),
    source VARCHAR(50) NOT NULL
```

```
);

-- Convert to hypertable partitioned by time
SELECT create_hypertable('weather_timeseries', 'time');

-- Create indices
CREATE INDEX idx_weather_timeseries_region_id ON
weather_timeseries(region_id);
```

Redis Schema (Caching)

Redis digunakan untuk caching dan penyimpanan data sementara:

1. **Session Storage**

```
Key: `session:{sessionId}`
Type: Hash
Fields:
  - userId: UUID pengguna
  - role: Role pengguna
  - permissions: JSON string of permissions
  - expiresAt: UNIX timestamp
TTL: 3600 (1 jam)
```

1. **Price Cache**

```
Key: `price:{commodityId}:{regionId}:latest`
Type: Hash
Fields:
  - price: Decimal value
  - timestamp: ISO date string
  - source: String
TTL: 300 (5 menit)
```

1. **Dashboard Cache**

```
Key: `dashboard:{userId}:{dashboardId}`
Type: String (JSON)
Value: Pre-rendered dashboard data
TTL: 600 (10 menit)
```

1. **Rate Limiting**

```
Key: `ratelimit:{ip}:{endpoint}`
Type: String
Value: Count of requests
TTL: 60 (1 menit)
```

1. **API Response Cache**

```
Key: `api:{endpoint}:{queryHash}`
Type: String (JSON)
Value: API response data
TTL: Bervariasi (60-900 detik)
```

🔌 API Design

RESTful API Endpoints

API menggunakan konvensi RESTful dan mendukung pagination, filtering, dan sorting.

User Management API

```
# Authentication
POST   /api/auth/login              - User login
POST   /api/auth/logout             - User logout
POST   /api/auth/refresh-token      - Refresh JWT token
POST   /api/auth/forgot-password    - Request password reset
POST   /api/auth/reset-password     - Reset password

# User Management
GET    /api/users                   - List users (admin only)
POST   /api/users                   - Create user (admin only)
GET    /api/users/:id               - Get user details
PUT    /api/users/:id               - Update user
DELETE /api/users/:id               - Delete user (admin only)

# Profile Management
GET    /api/profile                 - Get current user profile
PUT    /api/profile                 - Update current user profile
PUT    /api/profile/password        - Change password
GET    /api/profile/preferences     - Get user preferences
PUT    /api/profile/preferences     - Update user preferences

# Role Management
GET    /api/roles                   - List roles (admin only)
POST   /api/roles                   - Create role (admin only)
GET    /api/roles/:id               - Get role details
PUT    /api/roles/:id               - Update role (admin only)
DELETE /api/roles/:id               - Delete role (admin only)
GET    /api/roles/:id/permissions   - Get role permissions
PUT    /api/roles/:id/permissions   - Update role permissions
```

Commodity Data API

```
# Commodities
GET    /api/commodities             - List commodities
POST   /api/commodities             - Create commodity (admin only)
GET    /api/commodities/:id         - Get commodity details
PUT    /api/commodities/:id         - Update commodity (admin only)
DELETE /api/commodities/:id         - Delete commodity (admin only)

# Categories
GET    /api/commodity-categories                - List categories
POST   /api/commodity-categories                - Create category
(admin only)
GET    /api/commodity-categories/:id            - Get category
details
PUT    /api/commodity-categories/:id            - Update category
(admin only)
DELETE /api/commodity-categories/:id            - Delete category
(admin only)
GET    /api/commodity-categories/:id/commodities - Get commodities
in category

# Price Data
GET    /api/price-data              - List price data (with
filters)
POST   /api/price-data              - Submit price data
GET    /api/price-data/:id          - Get price data details
PUT    /api/price-data/:id          - Update price data (admin
only)
DELETE /api/price-data/:id          - Delete price data (admin
only)

# Price Analysis
GET    /api/price-analysis/trends               - Get price trends
GET    /api/price-analysis/comparison           - Compare prices
GET    /api/price-analysis/volatility           - Get price
volatility
```

```
GET    /api/price-analysis/seasonal             - Get seasonal
patterns
```

Regional Data API

```
# Regions
GET    /api/regions                 - List regions
POST   /api/regions                 - Create region (admin only)
GET    /api/regions/:id             - Get region details
PUT    /api/regions/:id             - Update region (admin only)
DELETE /api/regions/:id             - Delete region (admin only)
GET    /api/regions/:id/subregions  - Get subregions

# Geospatial
GET    /api/geo/commodities         - Get commodity data with
geolocation
GET    /api/geo/prices              - Get price data with
geolocation
GET    /api/geo/weather             - Get weather data with
geolocation
GET    /api/geo/production          - Get production data with
geolocation
```

Weather Data API

```
# Weather Data
GET    /api/weather-data            - List weather data (with
filters)
POST   /api/weather-data            - Submit weather data (admin
only)
GET    /api/weather-data/:id        - Get weather data details
PUT    /api/weather-data/:id        - Update weather data (admin
only)
DELETE /api/weather-data/:id        - Delete weather data (admin
only)

# Weather Forecasts
GET    /api/weather-forecasts       - Get weather forecasts
GET    /api/weather-forecasts/:regionId - Get forecasts for region
```

Analytics API

```
# Dashboard
GET    /api/dashboard               - Get dashboard data
GET    /api/dashboard/summary       - Get summary statistics
GET    /api/dashboard/widgets       - Get widget data
POST   /api/dashboard/widgets       - Create custom widget
PUT    /api/dashboard/widgets/:id   - Update widget
DELETE /api/dashboard/widgets/:id   - Delete widget

# Analytics
GET    /api/analytics/price-trends            - Get price trends
GET    /api/analytics/weather-impact          - Get weather impact
analysis
GET    /api/analytics/supply-chain            - Get supply chain
analytics
GET    /api/analytics/regional-comparison     - Get regional
comparison
GET    /api/analytics/market-insights         - Get market insights
```

Prediction API

```
# Price Predictions
GET    /api/predictions/prices              - Get price predictions
GET    /api/predictions/prices/:commodityId - Get predictions for
commodity
GET    /api/predictions/prices/:commodityId/:regionId - Get
regional predictions

# Anomaly Detection
GET    /api/anomalies                       - Get detected
anomalies
GET    /api/anomalies/:commodityId          - Get anomalies for
commodity
GET    /api/anomalies/:commodityId/:regionId - Get regional
anomalies

# Distribution Recommendations
GET    /api/recommendations/distribution     - Get distribution
recommendations
GET    /api/recommendations/inventory        - Get inventory
recommendations
GET    /api/recommendations/planting         - Get planting
recommendations
```

Alert API

```
# Alert Rules
GET    /api/alert-rules             - List alert rules
POST   /api/alert-rules             - Create alert rule
GET    /api/alert-rules/:id         - Get alert rule details
PUT    /api/alert-rules/:id         - Update alert rule
DELETE /api/alert-rules/:id         - Delete alert rule

# Alerts
GET    /api/alerts                  - List alerts
GET    /api/alerts/:id              - Get alert details
PUT    /api/alerts/:id/resolve      - Resolve alert
DELETE /api/alerts/:id              - Delete alert

# Notifications
GET    /api/notifications           - List notifications
GET    /api/notifications/:id       - Get notification details
PUT    /api/notifications/:id/read  - Mark notification as read
DELETE /api/notifications/:id       - Delete notification
PUT    /api/notifications/read-all  - Mark all notifications as
read
```

Reporting API

```
# Report Templates
GET    /api/report-templates        - List report templates
POST   /api/report-templates        - Create report template
GET    /api/report-templates/:id    - Get report template details
PUT    /api/report-templates/:id    - Update report template
DELETE /api/report-templates/:id    - Delete report template

# Reports
GET    /api/reports                 - List generated reports
POST   /api/reports/generate        - Generate report
GET    /api/reports/:id             - Get report details
GET    /api/reports/:id/download    - Download report
DELETE /api/reports/:id             - Delete report
```

Admin API

```
# System Settings
GET    /api/admin/settings          - Get system settings
PUT    /api/admin/settings          - Update system settings

# Data Sources
GET    /api/admin/data-sources      - List data sources
POST   /api/admin/data-sources      - Add data source
GET    /api/admin/data-sources/:id  - Get data source details
PUT    /api/admin/data-sources/:id  - Update data source
DELETE /api/admin/data-sources/:id  - Delete data source

# System Monitoring
GET    /api/admin/system-health     - Get system health
GET    /api/admin/logs              - Get system logs
GET    /api/admin/usage-stats       - Get usage statistics

# User Management (Admin)
GET    /api/admin/users             - Advanced user management
PUT    /api/admin/users/:id/role    - Change user role
PUT    /api/admin/users/:id/status  - Activate/deactivate user

# Audit Logs
GET    /api/admin/audit-logs        - View audit logs
GET    /api/admin/audit-logs/:id    - Get audit log details
```

GraphQL Schema

Sebagai komplemen RESTful API, sistem ini juga menyediakan GraphQL API untuk operasi kompleks dan data fetching yang fleksibel:

```
type Query {
  # User queries
  me: User
  user(id: ID!): User
  users(filter: UserFilter, pagination: PaginationInput):
UserConnection

  # Commodity queries
  commodity(id: ID!): Commodity
  commodities(filter: CommodityFilter, pagination:
PaginationInput): CommodityConnection
  commodityCategories: [CommodityCategory!]!

  # Price queries
  prices(
    commodityId: ID,
    regionId: ID,
    startDate: String,
    endDate: String,
    pagination: PaginationInput
  ): PriceDataConnection

  priceTimeseries(
    commodityId: ID!,
    regionId: ID!,
    startDate: String!,
    endDate: String!,
    interval: TimeInterval!
  ): [PriceTimeseriesPoint!]!

  # Weather queries
  weatherData(
    regionId: ID!,
    startDate: String!,
    endDate: String!
```

```
  ): [WeatherData!]!

  weatherForecast(regionId: ID!): [WeatherForecast!]!

  # Region queries
  region(id: ID!): Region
  regions(filter: RegionFilter, pagination: PaginationInput):
RegionConnection

  # Analytics
  priceTrends(
    commodityId: ID!,
    regionIds: [ID!],
    period: Period!
  ): [PriceTrend!]!

  priceComparison(
    commodityIds: [ID!]!,
    regionId: ID!,
    period: Period!
  ): [PriceComparison!]!

  weatherImpact(
    commodityId: ID!,
    regionId: ID!,
    period: Period!
  ): WeatherImpactAnalysis

  # Predictions
  pricePredictions(
    commodityId: ID!,
    regionId: ID!,
    daysAhead: Int!
  ): [PricePrediction!]!
```

```
  anomalies(
    commodityId: ID,
    regionId: ID,
    startDate: String,
    endDate: String,
    pagination: PaginationInput
  ): AnomalyConnection

  # Alerts
  alertRules(filter: AlertRuleFilter): [AlertRule!]!
  alerts(filter: AlertFilter, pagination: PaginationInput):
AlertConnection
  notifications(isRead: Boolean): [Notification!]!

  # Reports
  reportTemplates: [ReportTemplate!]!
  reports(pagination: PaginationInput): [GeneratedReport!]!

  # Dashboard
  dashboard: Dashboard
}

type Mutation {
  # Auth mutations
  login(email: String!, password: String!): AuthPayload
  logout: Boolean!
  refreshToken: AuthPayload
  forgotPassword(email: String!): Boolean!
  resetPassword(token: String!, newPassword: String!): Boolean!

  # User mutations
  updateProfile(input: UpdateProfileInput!): User
  changePassword(currentPassword: String!, newPassword: String!):
Boolean!
  updatePreferences(preferences: JSON!): User
```

```
  # Admin mutations (for authorized users)
  createUser(input: CreateUserInput!): User
  updateUser(id: ID!, input: UpdateUserInput!): User
  deleteUser(id: ID!): Boolean!
  updateUserRole(userId: ID!, roleId: ID!): User

  # Commodity mutations
  createCommodity(input: CommodityInput!): Commodity
  updateCommodity(id: ID!, input: CommodityInput!): Commodity
  deleteCommodity(id: ID!): Boolean!

  # Price data mutations
  submitPriceData(input: PriceDataInput!): PriceData
  updatePriceData(id: ID!, input: PriceDataInput!): PriceData
  verifyPriceData(id: ID!, isVerified: Boolean!): PriceData

  # Alert mutations
  createAlertRule(input: AlertRuleInput!): AlertRule
  updateAlertRule(id: ID!, input: AlertRuleInput!): AlertRule
  deleteAlertRule(id: ID!): Boolean!
  resolveAlert(id: ID!, notes: String): Alert
  markNotificationRead(id: ID!): Notification
  markAllNotificationsRead: Boolean!

  # Report mutations
  createReportTemplate(input: ReportTemplateInput!): ReportTemplate
  generateReport(templateId: ID!, parameters: JSON!):
GeneratedReport
}

# Subscription for real-time updates
type Subscription {
  priceUpdated(commodityId: ID, regionId: ID): PriceData
  alertCreated: Alert
```

```
  notificationCreated: Notification
}
```

WebSocket Implementation

WebSocket digunakan untuk real-time updates dan notifikasi:

WebSocket Endpoints

```
/ws/prices            - Real-time price updates
/ws/alerts            - Real-time alerts and notifications
/ws/dashboard         - Real-time dashboard updates
/ws/analytics         - Real-time analytics updates
```

WebSocket Message Format

```
{
  "type": "MESSAGE_TYPE",
  "payload": {
    // Message-specific data
  },
  "timestamp": "ISO_TIMESTAMP"
}
```

Message Types

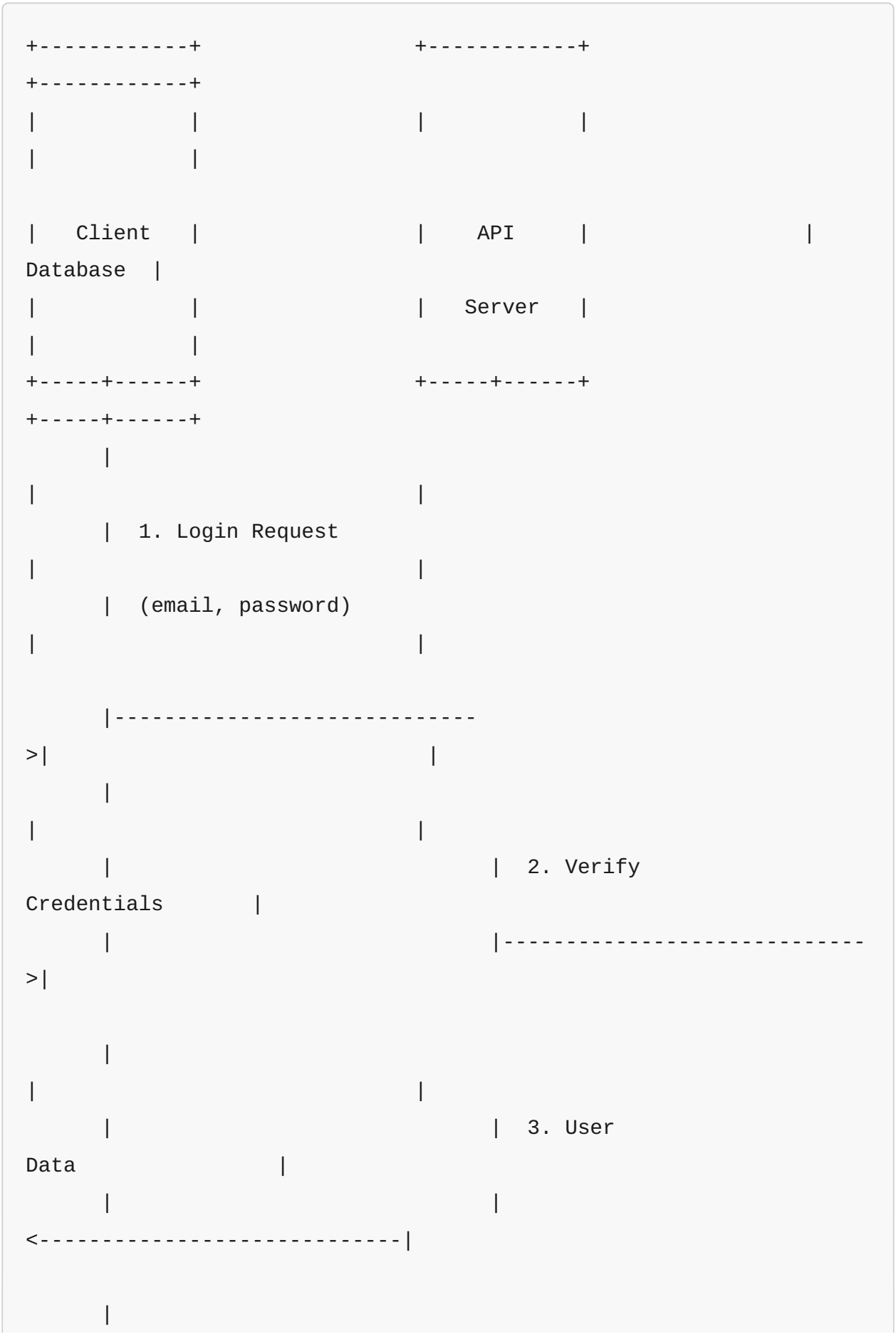
```
# Price Updates
PRICE_UPDATED          - New price data available
PRICE_ANOMALY          - Price anomaly detected
PRICE_TREND            - Significant price trend detected

# Alerts
ALERT_TRIGGERED        - New alert triggered
ALERT_RESOLVED         - Alert resolved

# Notifications
NOTIFICATION_CREATED   - New notification
SYSTEM_NOTIFICATION    - System notification

# Dashboard
DASHBOARD_UPDATE       - Dashboard data updated
WIDGET_UPDATE          - Specific widget updated

# System
CONNECTION_ACK         - Connection acknowledged
PING/PONG              - Keep-alive messages
ERROR                  - Error message
```

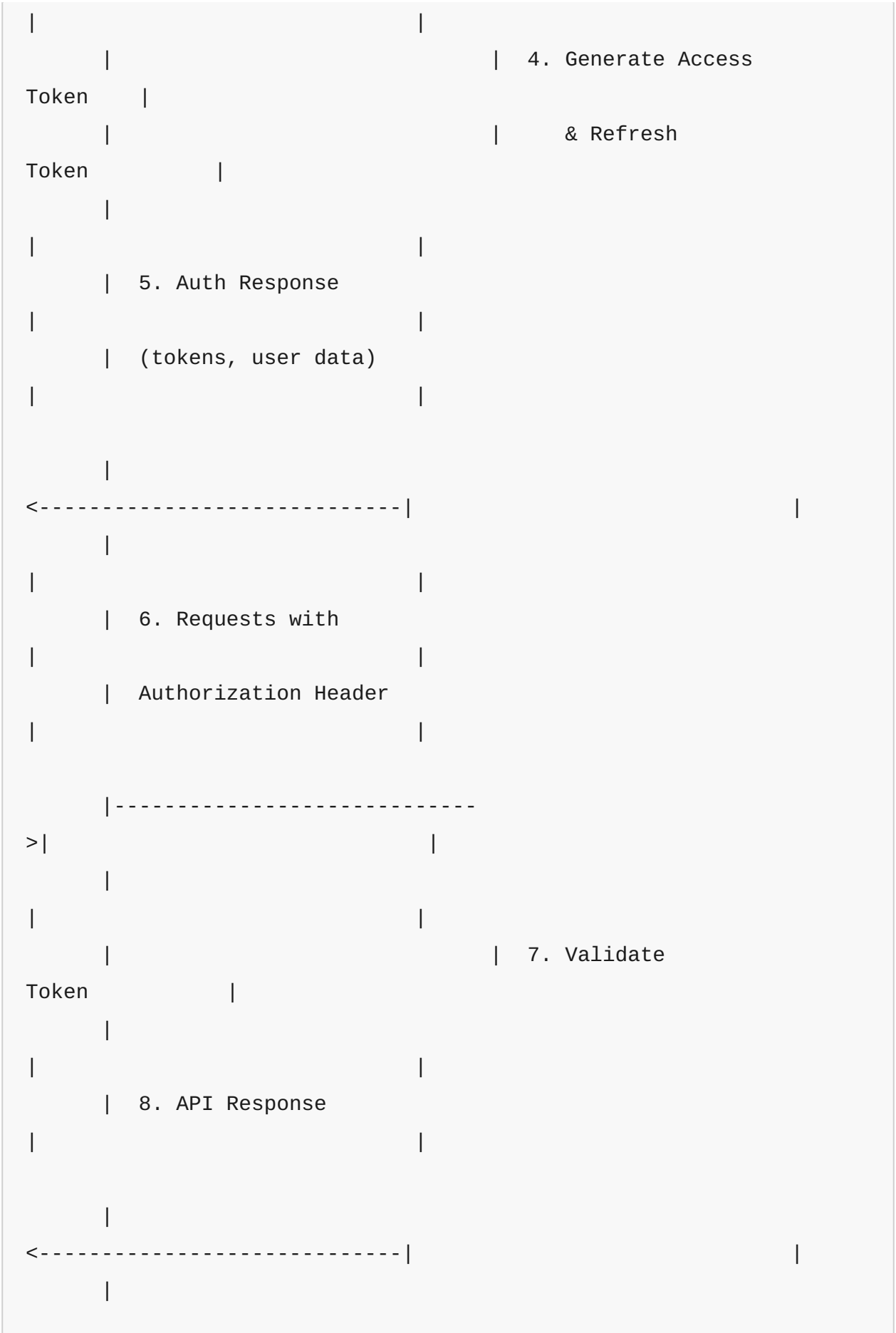
🔐 Authentication & Authorization

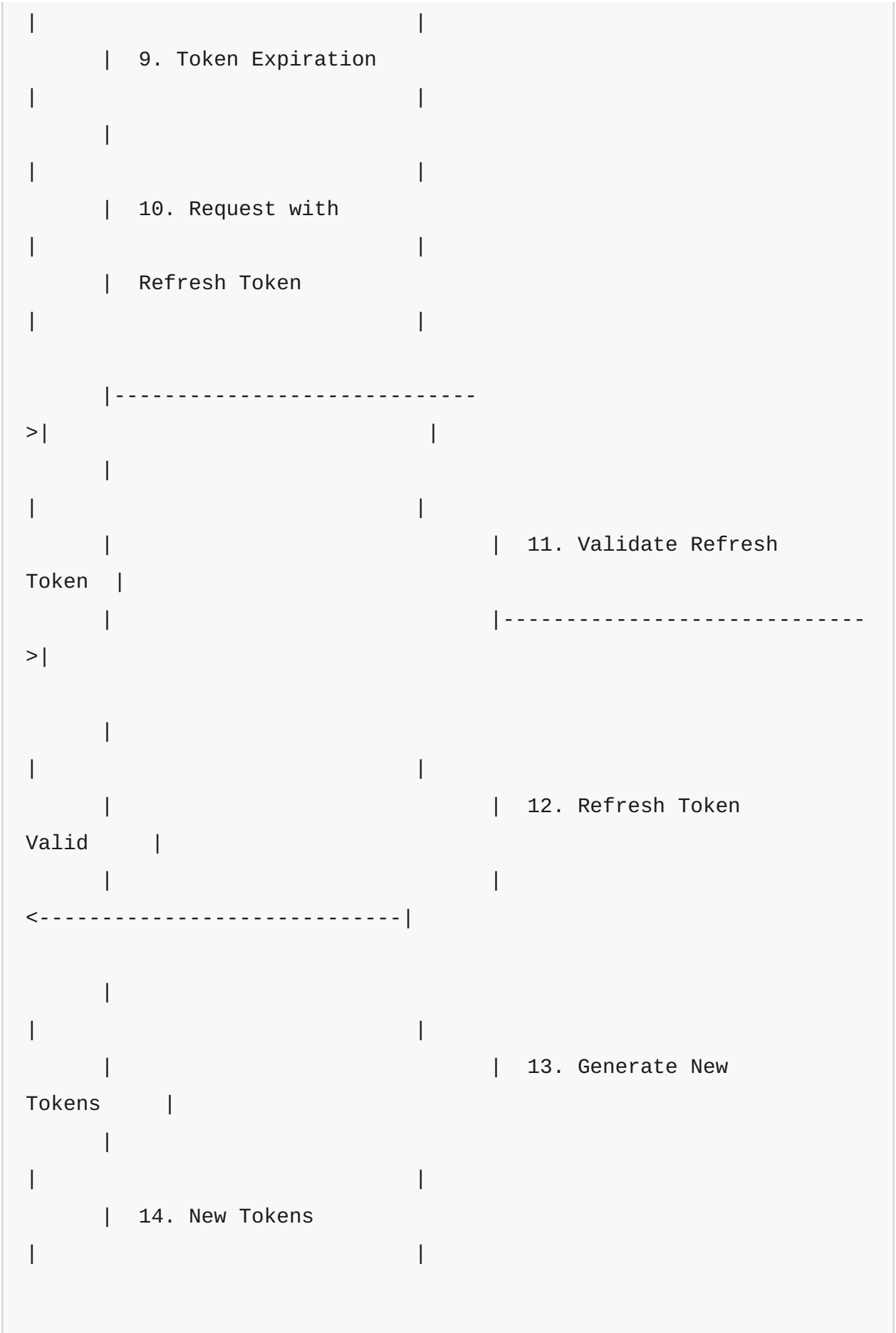
Authentication System

Authentication menggunakan JWT (JSON Web Tokens) dengan refresh token mechanism dan multi-factor authentication (opsional).

Authentication Flow


Client
API Server
Database
1. Login Request
(email, password)
2. Verify Credentials
3. User Data


4. Generate Access Token & Refresh Token
5. Auth Response (tokens, user data)
6. Requests with Authorization Header
7. Validate Token
8. API Response


9. Token Expiration
10. Request with
Refresh Token
11. Validate Refresh Token
12. Refresh Token Valid
13. Generate New Tokens
14. New Tokens

```
       |
<----------------------------|                                |
       |
|                              |
```

Token Management

1. **Access Token**
   - Short-lived JWT (15-30 menit)
   - Contains: User ID, role, essential permissions
   - Signed dengan RS256 algorithm
   - Stored in memory (no localStorage)
2. **Refresh Token**
   - Longer-lived token (7 hari)
   - Stored sebagai HTTP-only secure cookie
   - Used untuk mendapatkan new access token
   - Rotated pada setiap penggunaan
3. **Security Measures**
   - CSRF protection dengan custom headers
   - Token blacklisting untuk revoked tokens
   - Rate limiting untuk authentication endpoints
   - IP-based suspicious activity detection

Authorization Framework

Role-Based Access Control (RBAC) dengan permission granular.

User Roles

1. **Admin**
   - Full system access
   - User management
   - System configuration
   - Data validation dan verification

2. **Regulator**
   - View all commodity data
   - Generate advanced reports
   - Access prediction models
   - Configure alert rules
3. **Distributor**
   - View market data dan price trends
   - Access to distribution recommendations
   - Submit supply chain data
   - Limited data entry
4. **Farmer**
   - View basic market data
   - Submit production dan price data
   - Access weather forecasts
   - Receive market alerts

Permission Structure

```
resource:[action]

Examples:
- users:read
- users:write
- commodities:read
- commodities:write
- prices:read
- prices:write
- prices:verify
- analytics:read
- predictions:read
- reports:generate
- alerts:manage
```

Permission Assignment

Permissions diassign ke roles, dan roles diassign ke users. Custom permissions dapat diassign langsung ke specific users.

Access Control Implementation

1. **API Level**
   - Middleware authorization check
   - Route-specific permission requirements
   - GraphQL directive-based permissions
2. **Service Level**
   - Service method authorization
   - Domain-specific permission checks
3. **UI Level**
   - Conditional rendering berdasarkan permissions
   - Disabled actions untuk unauthorized operations

🔒 Security Framework

Security Layers

1. **Infrastructure Security**
   - Virtual Private Cloud (VPC) configuration
   - Network segmentation dan security groups
   - Firewall rules dan intrusion detection
   - DDoS protection
2. **Transport Security**
   - TLS 1.3 untuk semua connections
   - HTTPS enforcement
   - Strong cipher suites
   - HTTP security headers

3. **Application Security**
   - Input validation dan sanitization
   - Output encoding
   - SQL injection prevention
   - XSS protection
   - CSRF protection
4. **Authentication Security**
   - Secure password storage (bcrypt)
   - Brute force protection
   - Multi-factor authentication (optional)
   - Session management
5. **Data Security**
   - Encryption at rest
   - Encryption in transit
   - Database column-level encryption
   - Data masking untuk sensitive data

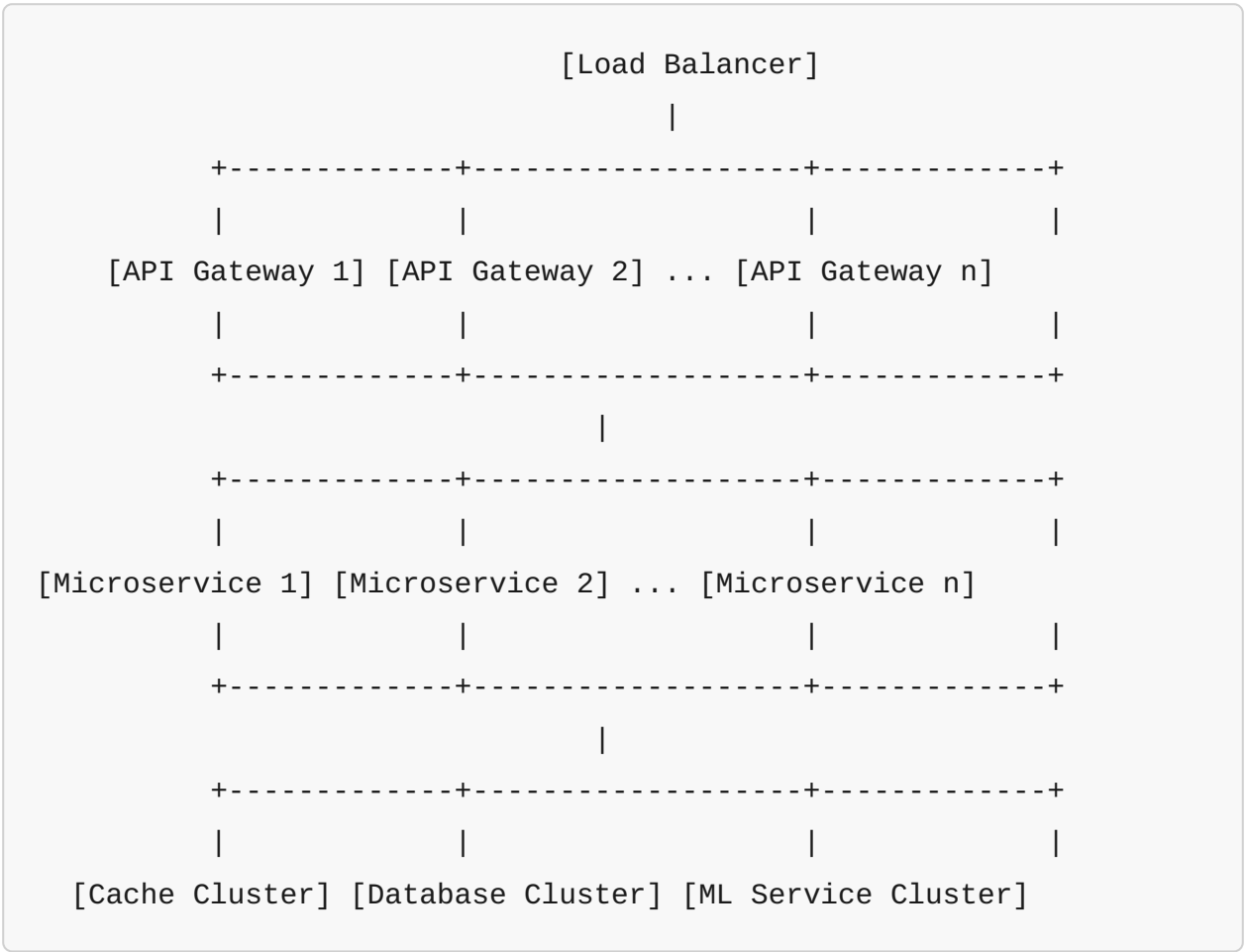
Security Features

1. **Access Control**
   - Role-based access control
   - Principle of least privilege
   - Permission-based authorization
   - IP-based access restrictions (optional)
2. **Audit dan Logging**
   - Comprehensive security logging
   - Audit trails for all sensitive operations
   - Log integrity protection
   - Real-time security monitoring
3. **API Security**
   - API key management
   - Rate limiting
   - Request validation
   - API versioning

4. **Data Privacy**
   - GDPR compliance
   - Data anonymization options
   - Privacy by design principles
   - User consent management
5. **Security Monitoring**
   - Real-time threat detection
   - Anomaly detection
   - Security alerting
   - Vulnerability scanning

📈 Scalability & Performance

Scalability Architecture



Horizontal Scaling

1. **Stateless Services**
   - API Gateway dan microservices didesain stateless
   - Load balancing distribusi beban merata
   - Auto-scaling berdasarkan CPU, memory, dan request metrics
2. **Database Scaling**
   - Read replicas untuk database queries
   - Connection pooling
   - Partitioning dan sharding strategies
   - Database proxy untuk load distribution
3. **Caching Tiers**
   - Multi-level caching strategy
   - Distributed Redis cluster
   - In-memory caching di application layer
   - HTTP caching dengan appropriate headers

Performance Optimization

1. **API Performance**
   - Response compression
   - Efficient serialization
   - Query optimization
   - Pagination dan hasil filtering
   - Partial response (GraphQL-based)
2. **Database Performance**
   - Indexed queries
   - Query optimization
   - Connection pooling
   - Data partitioning
   - Efficient JOIN operations

3. **Frontend Performance**
   - Code splitting
   - Lazy loading
   - Asset optimization
   - Client-side caching
   - Progressive Web App features
4. **Asynchronous Processing**
   - Background job processing
   - Message queues untuk resource-intensive tasks
   - Non-blocking I/O operations
   - Event-driven architecture

Caching Strategy

1. **Data Caching**
   - Time-based caching untuk static data
   - Version-based invalidation untuk dynamic data
   - Entity-based cache segmentation
   - Cache warming untuk critical data
2. **API Response Caching**
   - Cache berdasarkan user role dan request parameters
   - ETags untuk conditional requests
   - Cache invalidation events
   - Cache status monitoring
3. **Computed Results Caching**
   - Precomputed analytics results
   - Dashboard data caching
   - Cache invalidation pada data updates
   - Stale-while-revalidate pattern

🚀 Deployment Strategy

Containerization

Docker containers digunakan untuk semua services dengan standardized deployment process:

```
# Example Dockerfile for backend services
FROM node:20-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/node_modules ./node_modules
COPY package*.json ./
USER node
EXPOSE 3000
CMD ["node", "dist/main.js"]
```

CI/CD Pipeline

```
# Example CI/CD workflow
name: Build and Deploy

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main, develop]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '20'
      - name: Install dependencies
        run: npm ci
      - name: Run tests
        run: npm test
      - name: Run linting
        run: npm run lint

  build:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Build Docker image
        run: docker build -t komoditas-watch:${{ github.sha }} .
      - name: Push to container registry
        run: |
          docker tag komoditas-watch:<span class="math-inline"
```

```
style="display: inline;"><math xmlns="http://www.w3.org/1998/Math/
MathML" display="inline"><mrow><mrow><mrow><mi>g</mi><mi>i</
mi><mi>t</mi><mi>h</mi><mi>u</mi><mi>b</mi><mo>&#x0002E;</
mo><mi>s</mi><mi>h</mi><mi>a</mi></mrow></mrow><mi>r</mi><mi>e</
mi><mi>g</mi><mi>i</mi><mi>s</mi><mi>t</mi><mi>r</mi><mi>y</
mi><mo>&#x0002E;</mo><mi>e</mi><mi>x</mi><mi>a</mi><mi>m</
mi><mi>p</mi><mi>l</mi><mi>e</mi><mo>&#x0002E;</mo><mi>c</
mi><mi>o</mi><mi>m</mi><mo>&#x0002F;</mo><mi>k</mi><mi>o</
mi><mi>m</mi><mi>o</mi><mi>d</mi><mi>i</mi><mi>t</mi><mi>a</
mi><mi>s</mi><mo>&#x02212;</mo><mi>w</mi><mi>a</mi><mi>t</
mi><mi>c</mi><mi>h</mi><mi>:</mi></mrow></math></
span>{{ github.sha }}
          docker push registry.example.com/komoditas-watch:$
{{ github.sha }}

  deploy:
    needs: build
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main' || github.ref == 'refs/
heads/develop'
    steps:
      - name: Deploy to environment
        run: |
          if [[ $GITHUB_REF == 'refs/heads/main' ]]; then
            ENVIRONMENT=production
          else
            ENVIRONMENT=staging
          fi
          # Deploy using infrastructure as code tool
          terraform apply -var="environment=<span class="math-
inline" style="display: inline;"><math xmlns="http://www.w3.org/
1998/Math/MathML" display="inline"><mrow><mi>E</mi><mi>N</
mi><mi>V</mi><mi>I</mi><mi>R</mi><mi>O</mi><mi>N</mi><mi>M</
mi><mi>E</mi><mi>N</mi><mi>T</mi><mi>"</mi><mo>&#x02212;</
mo><mi>v</mi><mi>a</mi><mi>r</mi><mo>&#x0003D;</mo><mi>"</
```

```
mi><mi>i</mi><mi>m</mi><mi>a</mi><mi>g</mi><msub><mi>e</mi><mi>t</
mi></msub><mi>a</mi><mi>g</mi><mo>&#x0003D;</mo></mrow></math></
span>{{ github.sha }}"
```

Environment Strategy

1. **Development Environment**
   - Local development dengan Docker Compose
   - Mock services untuk external dependencies
   - Hot reloading untuk code changes
   - Development database
2. **Staging Environment**
   - Identical dengan production infrastructure
   - Synthetic data dan test data
   - Integration testing
   - Performance testing
   - Security testing
3. **Production Environment**
   - Multiple regions deployment
   - Blue-green deployment strategy
   - Automated rollbacks
   - Production monitoring
   - Disaster recovery plan

Release Management

1. **Version Control**
   - Feature branches
   - Pull request workflow
   - Code review requirements
   - Version tagging

2. **Release Process**
   - Semantic versioning
   - Release notes generation
   - Changelog maintenance
   - Automated deployment
3. **Rollback Strategy**
   - Automated rollback on failure
   - Previous version maintenance
   - Database migration rollback plans
   - State recovery procedures

🖥️ Infrastructure Requirements

Production Environment

Compute Resources

1. **API Gateway dan Backend Services**
   - **Instances**: Minimum 3 nodes per service
   - **Compute**: 4 vCPU, 8GB RAM per node
   - **Storage**: 50GB SSD per node
   - **Auto-scaling**: Trigger pada 70% CPU utilization
2. **Database Cluster**
   - **Primary**: 8 vCPU, 32GB RAM
   - **Read Replicas**: 3 instances, 4 vCPU, 16GB RAM each
   - **Storage**: 1TB SSD with auto-scaling
   - **Backup**: Daily snapshots, point-in-time recovery
3. **Redis Cluster**
   - **Instances**: 3-node cluster
   - **Compute**: 2 vCPU, 8GB RAM per node
   - **Storage**: In-memory
   - **Persistence**: RDB snapshots + AOF logs

4. **ML Service**
   - **Instances**: 2 nodes (GPU enabled)
   - **Compute**: 8 vCPU, 32GB RAM, 1 GPU
   - **Storage**: 100GB SSD
   - **Scaling**: Manual scaling for training, auto-scaling for inference
5. **Web/Frontend**
   - **CDN**: Global content delivery network
   - **Static Hosting**: S3 or equivalent
   - **Edge Functions**: For dynamic content

Network Requirements

1. **Bandwidth**
   - **External**: 100Mbps minimum
   - **Internal**: 1Gbps between services
   - **Burst Capacity**: 500Mbps for peak loads
2. **Latency Requirements**
   - **API Response**: <200ms average
   - **Database Queries**: <50ms average
   - **Service Communication**: <20ms average
3. **Security Requirements**
   - **WAF**: Web Application Firewall
   - **DDoS Protection**: Advanced mitigation
   - **VPN**: For administrative access
   - **Network Isolation**: Private subnets for databases

Storage Requirements

1. **Database Storage**
   - **Type**: High-performance SSD
   - **Initial Size**: 1TB
   - **Growth Rate**: Estimated 20GB/month
   - **Backup Storage**: 5TB for backups

2. **Object Storage**
   - **Type**: S3-compatible
   - **Initial Size**: 100GB
   - **Use Cases**: Reports, exports, uploads
   - **Lifecycle Policies**: Archive after 90 days
3. **Log Storage**
   - **Type**: Centralized logging
   - **Retention**: 30 days hot, 1 year cold
   - **Size**: Approximately 5GB/day

Staging Environment

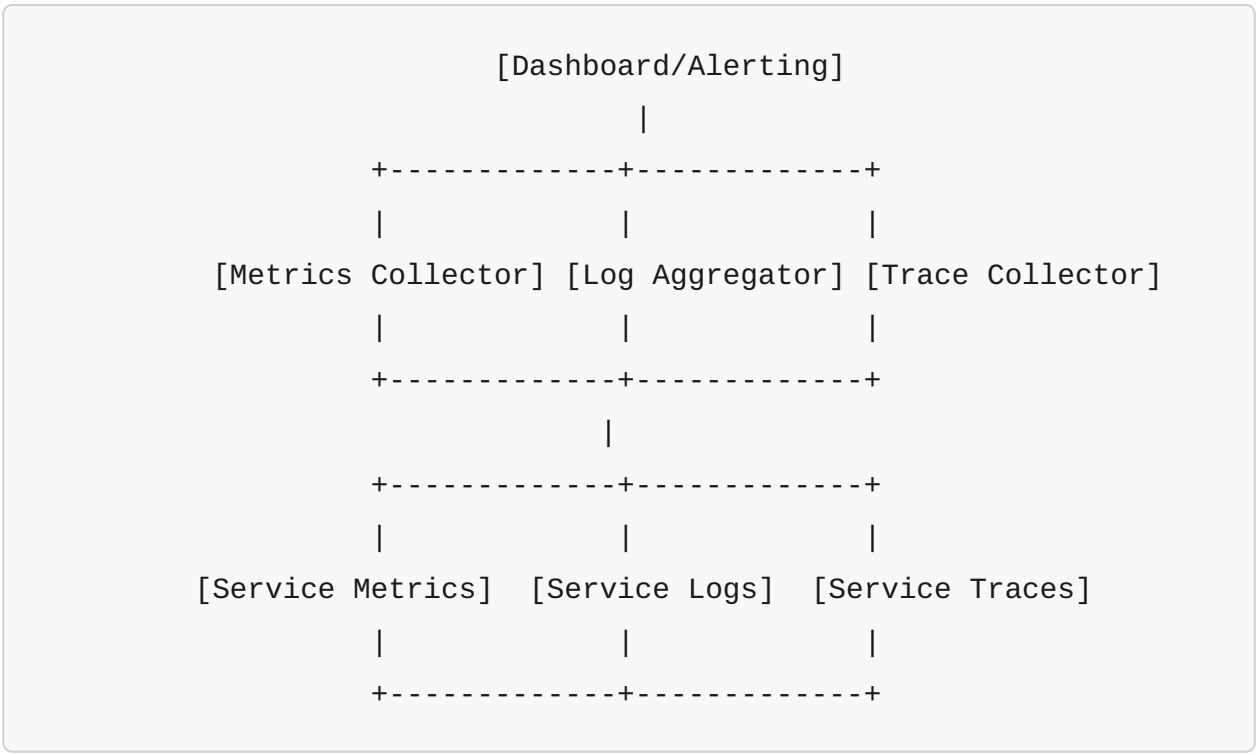
- 50% of production resources
- Same architecture but scaled down
- Full database copies with synthetic data

Development Environment

- Docker Compose configuration
- Local development setup
- Mock services for external dependencies

🔍 Monitoring & Error Handling

Monitoring Framework



Metrics Monitoring

1. **System Metrics**
   - CPU, memory, disk usage
   - Network throughput dan latency
   - Container health
   - Load average
2. **Application Metrics**
   - Request count dan rate
   - Response time (avg, p95, p99)
   - Error rate
   - Concurrent users

3. **Business Metrics**
   - Active users
   - Data submission rate
   - Feature usage
   - User engagement
4. **External Dependency Metrics**
   - API response times
   - Success/failure rates
   - Availability
   - Rate limit usage

Log Management

1. **Log Collection**
   - Centralized log aggregation
   - Structured logging format (JSON)
   - Log tagging dan correlation
   - Multi-level logging (debug, info, warn, error)
2. **Log Analysis**
   - Real-time log search
   - Log-based alerting
   - Pattern recognition
   - Anomaly detection
3. **Log Retention**
   - Hot storage: 30 days
   - Cold storage: 1 year
   - Compliance archives: 7 years

Error Handling

1. **Error Categorization**
   - System errors
   - Application errors
   - Validation errors
   - External dependency errors

2. **Error Handling Strategy**
   - Graceful degradation
   - Retry mechanisms with exponential backoff
   - Circuit breakers untuk external dependencies
   - Fallback mechanisms
3. **Error Reporting**
   - Real-time error notifications
   - Error aggregation dan grouping
   - Root cause analysis tools
   - Error resolution tracking

Alerting Strategy

1. **Alert Tiers**
   - **Critical**: Immediate response required (24/7)
   - **High**: Same business day response
   - **Medium**: Next business day response
   - **Low**: Scheduled fix
2. **Alert Channels**
   - Email notifications
   - SMS for critical alerts
   - Integration dengan incident management system
   - Dashboard notifications
3. **Alert Triggers**
   - Error rate thresholds
   - Latency thresholds
   - System resource thresholds
   - Business KPI thresholds
4. **Alert Management**
   - Alert aggregation
   - Alert escalation
   - On-call rotation
   - Post-mortem process

🧯 Disaster Recovery Plan

Backup Strategy

1. **Database Backups**
   - Full daily backups
   - Incremental hourly backups
   - Point-in-time recovery capability
   - Multi-region backup replication
2. **Application State**
   - Configuration backups
   - User data backups
   - File storage backups
   - Encryption key backups (secure vault)
3. **Backup Testing**
   - Regular restore testing
   - Backup validation
   - Recovery time objective (RTO) verification

Disaster Recovery Scenarios

1. **Single Service Failure**
   - Automatic failover to redundant instances
   - Auto-scaling to replace failed instances
   - Circuit breaking for dependent services
2. **Database Failure**
   - Automatic failover to replica
   - Replica promotion
   - Read-replica redistribution
3. **Region Failure**
   - Cross-region failover
   - DNS-based traffic rerouting
   - Data synchronization resumption

4. **Complete Outage**
   - Full system restore from backups
   - Prioritized service restoration
   - Incremental capacity recovery

Recovery Time Objectives

1. **Tier 1 Services** (Critical)
   - RTO: 1 hour
   - RPO: 5 minutes
   - Examples: API Gateway, Auth Service, Core Data Services
2. **Tier 2 Services** (Important)
   - RTO: 4 hours
   - RPO: 1 hour
   - Examples: Analytics Service, Notification Service
3. **Tier 3 Services** (Non-critical)
   - RTO: 24 hours
   - RPO: 24 hours
   - Examples: Reporting Service, Admin Service

Business Continuity

1. **Degraded Mode Operation**
   - Essential functionality preservation
   - Cached data usage
   - Read-only mode capabilities
   - Offline capabilities
2. **Communication Plan**
   - User notification procedures
   - Stakeholder communication templates
   - Status page updates
   - Escalation path

3. **Recovery Procedures**
   - Step-by-step restoration guide
   - Responsibility matrix
   - Verification checklists
   - Post-recovery validation

📅 Development Roadmap

Phase 1: Foundation (Month 1-2)

1. **Core Infrastructure Setup**
   - Base architecture implementation
   - CI/CD pipeline setup
   - Development environment
   - Database schema implementation
2. **API Integration**
   - BPS Web API integration
   - BMKG XML Parser development
   - Global Commodities API integration
   - Data normalization layer
3. **Basic Frontend**
   - Authentication system
   - Basic dashboard
   - Commodity listing
   - Price data visualization

Phase 2: Enhanced Features (Month 3-4)

1. **Advanced Analytics**
   - Time series analysis
   - Price trend visualization
   - Regional comparison
   - Weather impact analysis

2. **ML Pipeline**
   - Data preprocessing pipeline
   - Basic forecasting models
   - Model training workflow
   - Prediction API
3. **Web Scraping**
   - Panel Harga scraper
   - Data extraction pipeline
   - Data validation
   - Integration with main system
4. **User Management**
   - Role-based access control
   - User profile management
   - Permission system
   - Admin interface

Phase 3: Optimization & Advanced Features (Month 5-6)

1. **Advanced ML Models**
   - LSTM model implementation
   - Anomaly detection system
   - Model performance optimization
   - Automated retraining pipeline
2. **Real-time Features**
   - WebSocket implementation
   - Real-time dashboard updates
   - Real-time notifications
   - Alert system
3. **Mobile Responsiveness**
   - Mobile optimization
   - Offline capabilities
   - Progressive Web App features
   - Mobile-specific UI enhancements

4. **Report Generation**
   - Report template system
   - PDF generation
   - Excel export
   - Scheduled reports

Phase 4: Production Readiness (Month 7-8)

1. **Performance Optimization**
   - Load testing
   - Performance tuning
   - Caching optimization
   - Database optimization
2. **Security Hardening**
   - Security audit
   - Penetration testing
   - Vulnerability assessment
   - Security documentation
3. **Documentation**
   - API documentation
   - User documentation
   - Admin documentation
   - Developer documentation
4. **Deployment**
   - Staging environment setup
   - Production environment setup
   - Monitoring configuration
   - Backup and recovery testing

📚 Appendix

Technology Stack Details

Frontend

- **Framework**: React.js 18.x
- **State Management**: React Context API + React Query
- **Styling**: TailwindCSS 3.x
- **UI Components**: Custom component library
- **Data Visualization**: Recharts + ECharts + react-leaflet
- **Build Tool**: Vite
- **Testing**: Jest + React Testing Library

Backend

- **Runtime**: Node.js 20.x
- **Framework**: Express.js 4.x
- **API Documentation**: Swagger/OpenAPI
- **Validation**: Joi/Zod
- **Authentication**: Passport.js + JWT
- **ORM**: Prisma 5.x
- **Testing**: Jest + Supertest

Database

- **Primary Database**: PostgreSQL 15.x
- **Time Series Extension**: TimescaleDB
- **Caching**: Redis 7.x
- **Search**: PostgreSQL Full Text Search

ML / Data Science

- **Framework**: Python 3.11 + FastAPI
- **ML Libraries**: Scikit-learn, TensorFlow, Prophet
- **Data Processing**: Pandas, NumPy
- **Visualization**: Matplotlib, Seaborn

DevOps

- **Containerization**: Docker
- **Orchestration**: Kubernetes
- **CI/CD**: GitHub Actions
- **Infrastructure as Code**: Terraform
- **Monitoring**: Prometheus + Grafana
- **Logging**: ELK Stack (Elasticsearch, Logstash, Kibana)

API Integration Details

1. BPS Web API

Base URL: https://webapi.bps.go.id/v1/api/

Authentication:
- API Key dalam query parameter
- Key didapatkan melalui registrasi di webapi.bps.go.id/developer

Main Endpoints:
- `/list/model/data/domain/{domain}/var/{var_id}/key/{api_key}` - Dynamic Data
- `/dataexim/sumber/{source}/periode/{period}/kodehs/{hs_code}/key/{api_key}` - Foreign Trade Data

Key Variables for Commodity Prices:
- Consumer Price Index (ID: 2212)
- Wholesale Price Index (ID: 2498)
- Food Inflation Index (ID: 1890)

Implementation Notes:
- Implement caching untuk mengurangi API calls
- Setup circuit breaker untuk handle downtime
- Transform data responses ke standard format
- Log semua API interactions untuk debugging

2. BMKG Open Data

Base URL: https://data.bmkg.go.id/DataMKG/MEWS/DigitalForecast/

Format: XML files per province

Access Pattern:
- `DigitalForecast-{Province}.xml`
- Example: `DigitalForecast-JawaBarat.xml`

Implementation Notes:
- XML parsing menggunakan efficient parser
- Extract only relevant fields (temperature, humidity, rainfall)
- Map weather parameters to agricultural impact factors
- Store parsed data in time series database

3. Global Commodities API

Integration Type: Client library

Commodities Covered:
- WHEAT, CORN, SUGAR, COFFEE, COCOA, etc.
- Total: 13 commodities

Currencies Supported: 158 including IDR

Implementation Notes:
- Use as reference for global market prices
- Correlate global prices with local prices
- Implement data normalization for consistent units

4. Panel Harga Scraper

Target Site: https://panelharga.badanpangan.go.id/

Scraping Approach:
- Scheduled scraping (daily)
- HTML parsing with cheerio/beautiful soup
- Data extraction dan normalization
- Proxy rotation to avoid blocking

Implementation Notes:
- Implement rate limiting to be respectful
- Include proper user-agent identification
- Setup monitoring for scraper health
- Validate extracted data for consistency

Database Sizing Estimates

Estimated Data Growth:

1. **Price Data**:
   - 8 commodities × 549 regions × daily records = 4,392 records/day
   - 1.6 million records/year
   - ~1GB/year (with indexes)
2. **Weather Data**:
   - 34 provinces × 4 records/day = 136 records/day
   - ~50,000 records/year
   - ~250MB/year (with indexes)
3. **User Data**:
   - Estimated 1,000 users
   - Low growth rate
   - ~10MB total
4. **Analytics & Predictions**:
   - ~100,000 prediction records/month
   - ~50,000 anomaly records/month
   - ~5GB/year

Total Estimated Size:
- Year 1: ~10GB
- Year 3: ~35GB
- Year 5: ~60GB

Internationalization Support

The system supports multiple languages with focus on Bahasa Indonesia and English:

1. **Frontend**:
   - i18next for translation management
   - Language selection in user preferences
   - Language detection based on browser settings
2. **Backend**:
   - Localized error messages
   - Date/time formatting based on locale
   - Currency formatting based on locale
3. **Content**:
   - Commodity names in multiple languages
   - Region names in multiple languages
   - UI elements fully translated

API Rate Limiting

1. **Public API**:
   - 60 requests/minute per IP
   - 1,000 requests/day per IP
2. **Authenticated API**:
   - 300 requests/minute per user
   - 10,000 requests/day per user
3. **Admin API**:
   - 600 requests/minute per admin
   - 50,000 requests/day per admin
4. **Rate Limit Headers**:
   - X-RateLimit-Limit
   - X-RateLimit-Remaining
   - X-RateLimit-Reset

Compliance Considerations

1. **Data Privacy**:
   - GDPR compliance for any EU users
   - PDP (Personal Data Protection) Indonesia compliance
   - Data minimization principles
   - User consent management
2. **Security Compliance**:
   - OWASP security best practices
   - Regular security audits
   - Vulnerability scanning
   - Penetration testing
3. **Accessibility**:
   - WCAG 2.1 AA compliance
   - Keyboard navigation
   - Screen reader support
   - Color contrast requirements