ClinicMaster Kafka Multi-Facility Data Streaming

Overview

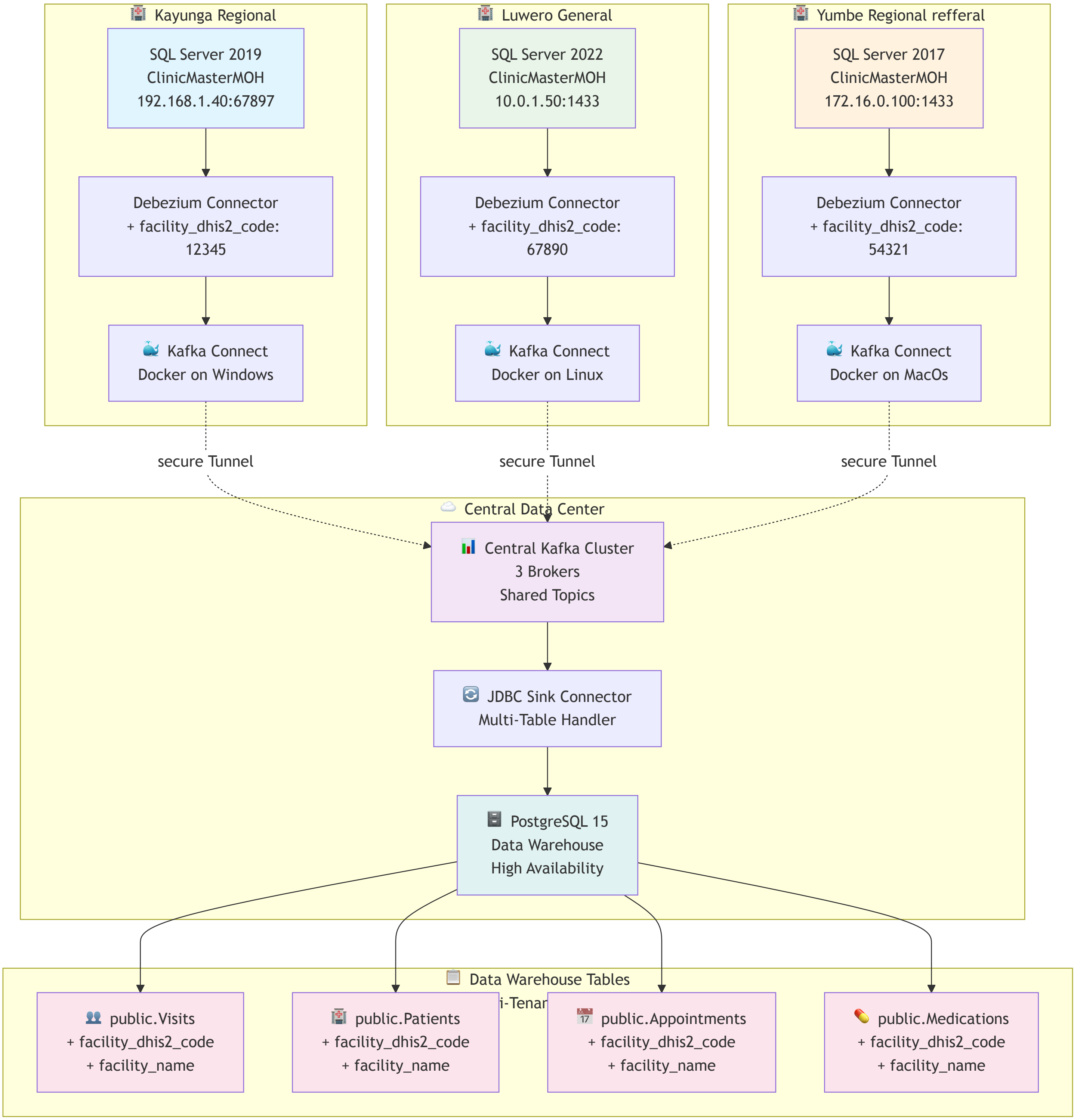
This document outlines a scalable, Docker-based Change Data Capture (CDC) pipeline that streams healthcare data from multiple facilities to a central data warehouse. Each facility runs a lightweight Kafka Connect container that captures changes from their local SQL Server database and streams them to shared Kafka topics, where a central JDBC sink replicates the data into PostgreSQL.

Architecture

Data Flow

1. **Facility-Level Capture**: Each facility deploys a Docker container with Kafka Connect and Debezium
2. **Central Aggregation**: All facilities stream to shared Kafka topics with facility metadata
3. **Data Warehouse**: A single JDBC sink replicates all data into PostgreSQL with facility isolation

System Diagram



Key Design Principles

Shared Topics with Facility Metadata

- **Single topic per table** across all facilities (e.g., `cm-sqlserver.ClinicMasterMOH.dbo.Visits`)

- **Facility identification** via `facility_dhis2_code` and `facility_name` in every record
- **Collision-free upserts** using composite keys that include facility code

Docker-Based Deployment

- **One container per facility** with minimal configuration
- **Environment-driven setup** via `.env` files
- **Auto-connector creation** on container startup

Error Resilience

- **Dead Letter Queue (DLQ)** for failed records. A Dead Letter Queue (DLQ) is a safety mechanism in streaming systems that isolates failed messages (e.g., due to schema errors or invalid data) after retries, routing them to a separate topic/queue. This keeps the main pipeline flowing uninterrupted while preserving "dead" records for later debugging, reprocessing, or analysis—preventing data loss without blocking consumers.
- **Error tolerance** to keep connectors running
- **Auto-table creation** and schema evolution

Source Connector Configuration

Each facility runs an identical Debezium SQL Server connector with facility-specific metadata injection:

Core Settings

- **Database**: Local SQL Server (private network)
- **Tables**: All `dbo.*` tables via regex
- **Topic Prefix**: `clinicmasterwarehouse-sqlserver` (shared across facilities)
- **Snapshot Mode**: `when_needed` for efficiency
- **Heartbeat**: 30-second intervals for low-traffic databases

Facility Metadata Injection

```
"transforms": "unwrap,addFacilityCodeValue,addFacilityNameValue,addFacilityCodeKey",
"transforms.addFacilityCodeValue.type": "org.apache.kafka.connect.transforms.InsertFiel
"transforms.addFacilityCodeValue.static.field": "facility_dhis2_code",
"transforms.addFacilityCodeValue.static.value": "__FACILITY_DHIS2_CODE__",
"transforms.addFacilityNameValue.type": "org.apache.kafka.connect.transforms.InsertFiel
"transforms.addFacilityNameValue.static.field": "facility_name",
"transforms.addFacilityNameValue.static.value": "__FACILITY_NAME__",
"transforms.addFacilityCodeKey.type": "org.apache.kafka.connect.transforms.InsertField$
"transforms.addFacilityCodeKey.static.field": "facility_dhis2_code",
"transforms.addFacilityCodeKey.static.value": "__FACILITY_DHIS2_CODE__"
```

Resulting Event Structure

```
{
  "key": {"id": 123, "facility_dhis2_code": "12345"},
  "value": {
    "id": 123,
    "patient_name": "Test Patient",
    "visit_date": "2024-01-15",
    "facility_dhis2_code": "12345",
    "facility_name": "Kayunga Hospital"
  }
}
```

Sink Connector Configuration

A single JDBC sink handles all facilities with automatic table creation and error handling:

Core Settings

- **Topics**: Regex pattern `clinicmasterwarehouse-sqlserver\\..*\\.dbo\\..*`
- **Target**: PostgreSQL data warehouse
- **PK Mode**: `record_key` (uses composite key with facility code)
- **Auto-Create**: Enabled for new tables
- **Auto-Evolve**: Enabled for schema changes

Error Handling

```
"errors.tolerance": "all",
"errors.log.enable": "true",
"errors.log.include.messages": "true",
"errors.deadletterqueue.topic.name": "sink-dlq",
"errors.deadletterqueue.topic.replication.factor": "1"
```

Table Routing

- **Regex Router**: Extracts table name from topic
- **Format**: `public.${topic}` (e.g., `public.Visits`)
- **Facility Isolation**: Composite primary keys prevent cross-facility collisions

Deployment

Facility Setup

1. **Build and push** the Docker image centrally
2. **Deploy at facility**:

   ```
   cp .env.sample .env
   # Edit .env with facility-specific values
   docker compose up -d
   ```

3. **Verify**: Check connector status and topic creation

Environment Variables

```
# Facility identification
FACILITY_NAME=Kayunga Regional refferal
FACILITY_DHIS2_CODE=12345

# Database connection
DB_HOST=192.168.1.100
DB_PORT=1433
DB_USER=SA
DB_PASS=your_password
DB_NAME=ClinicMasterMOH

# Central Kafka
KAFKA_BOOTSTRAP_SERVERS=central-kafka-1.example.com:9093
KAFKA_USER=facility-user
KAFKA_PASS=your_kafka_password
```

Benefits

Scalability

- **Zero-configuration scaling**: Add facilities by deploying containers
- **Shared infrastructure**: Single sink handles all facilities
- **Auto-discovery**: New tables automatically create PostgreSQL tables

Operational Excellence

- **Docker-based**: Consistent deployment across facilities
- **Error resilience**: DLQ and error tolerance prevent data loss
- **Monitoring**: Centralized visibility into all facility data

Data Quality

- **Facility isolation**: Composite keys prevent data collisions
- **Schema evolution**: Automatic handling of table changes
- **Audit trail**: Facility metadata in every record

Monitoring and Maintenance

Key Metrics

- **Connector status** per facility
- **Topic lag** and throughput
- **DLQ growth** for error tracking
- **PostgreSQL table health**

Troubleshooting

- **Check connector logs**: `docker logs clinicmasterwarehouse`
- **Verify topics**: `curl http://localhost:8083/connectors`
- **Monitor DLQ**: Review failed records in `sink-dlq` topic
- **Database connectivity**: Ensure VPN and firewall rules