ETL Process Redesign

# Detailed Design

# Team 7351

## February 21, 2018

Noah Harper, Chris Polack, Chase Lewis, Bruce Yang, Zhijian Li

# 

# 

# 

# Table of Contents

# **1. Introduction**

# **2. System Architecture**

Static Elements

Dynamic Elements

# **3. Data Storage Design**

# **4. Component Detailed Design**

Static Elements

Dynamic Elements

# **5. UI Design**

# Figures & Tables

2.1 NBS Architecture Diagram

2.2 ETL Process Architecture Diagram

2.3 System Architecture Sequence Diagram

3.1 Relationship of main ODS tables

3.2 ODS Tables for storing additional supporting person information

3.3 ODS Person table for storing core information about a person

4.1 Static Component Diagram

4.2 Dynamic Component Diagram

# Acronyms

**CDC** - Centers for Disease Control and Prevention

**NEDSS** - National Electronic Disease Surveillance System

**NBS** - NEDSS Base System

**ETL** - Extract, Transform, Load

**ODS -** Operational Datastore

**RDB** - Reporting Database

# Introduction

The Centers for Disease Control and Prevention maintains an integrated information system called the National Electronic Disease Surveillance System. The NEDSS Base System allows states and health partners to enter disease data into a database accessible to health investigators, allows labs to upload reports to health departments, integrates multiple databases into a single integrated system, and allows partners to send and share information[[1]](#footnote-0). The system on which we are working is used in twenty-three states nationwide.

Currently, there is an Extract, Transform, and Load process implemented in the system that converts health department information into a format that more closely resembles traditional public health databases. The current ETL process is implemented via SAS, a proprietary statistical analysis software for which the CDC must pay licensing fees within each state. Our task is to aid the CDC in researching, designing, and implementing a new ETL process using open source software that can be easily built upon and used for all data dimensions of the NBS.

By writing an ETL process for the CDC that can be used without payment or third party restrictions, we will free up both monetary resources and contractionary restrictions for the organization. Our research will entail looking for an open source product that can be used instead of writing our own ETL tool specific to the CDC. Although the scope of our project is initially limited to only one area of the Operational Datastore, our solution must be generalizable so that it can be implemented system-wide in the future. Our research and development process is guided by important requirements including implementation simplicity, user adaptability, and computational power.

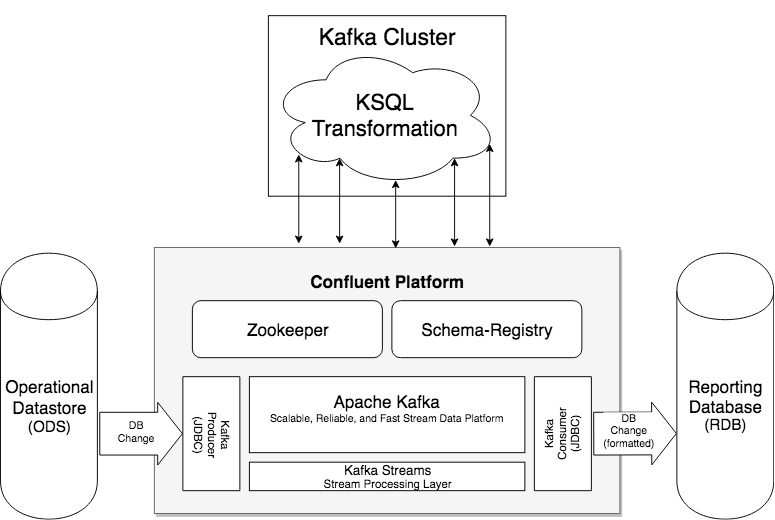
This document describes our project in detail through three primary aspects: system architecture, data storage design, and component design. Due to backend nature of this project, it was not within the scope of the task given to us by our client to design or edit a user interface for the ETL process.

# System Architecture

As the CDC’s NEDSS Base System is an existing system, many of its architectural components and decisions were outside the scope of our team’s project. Below are diagrams detailing this predefined architecture.

# 

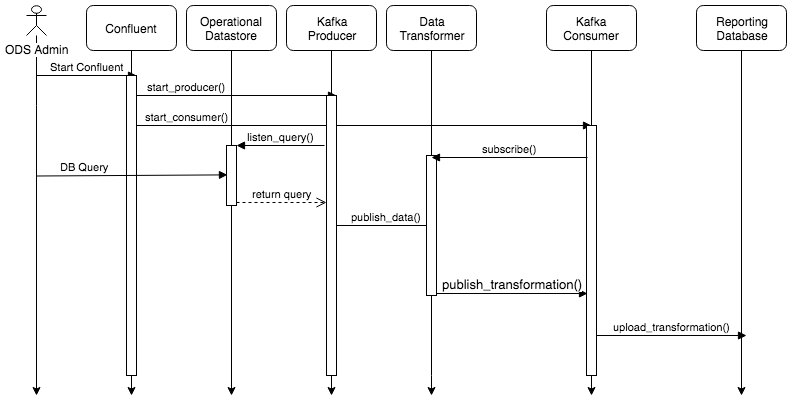
**Figure 2.1** NBS Architecture



**Figure 2.2** ETL Static Process Architecture

**Figure 2.1** is an overview of the NBS and it is included only to clarify the specific area of focus our team was tasked with in the context of the external architecture. The highlighted *Master ETL Batch Process* is this specific area of focus and below the NBS Architecture diagram is **Figure 2.2**, the structure of the new ETL Process as defined by our team. Here we have detailed how the open-source Confluent Platform - which will replace the old batch SAS ETL process - fits into the predefined data flow from the Operational Datastore to the Reporting Database. This architecture diagram also details the relevant components of Confluent that will be used by our ETL Process, including prepackaged software such as Kafka, Zookeeper, and the Schema Registry. In addition to the prepackaged software, the diagram details the Kafka Connectors (i.e. the Producer and the Consumer) and a representation of the Kafka Cluster that will make the relevant KSQL transformations to the data passing from the ODS to the RDB. The Connectors and the Kafka Cluster transformations are the elements from which the bulk of our team’s workload is sourced.

As mentioned above, the primary architectural decisions specific to this implementation were predefined by the CDC. The components identified in **Figure 2.2** were modeled after existing Confluent programs that involve data flow. The producer and the consumer are necessary for the publication of and subscription to information from the ODS to the Kafka Cluster and ultimately to the RDB. The KSQL Transformations must occur within the cluster, as that is where the information is stored once pulled from the ODS.



**Figure 2.3** ETL Dynamic Process Architecture

**Figure 2.3** is a sequence diagram that provides a sequential outline of the process we have designed for our open-source Kafka solution. The process begins when an ODS Administrator starts Confluent, which in turn begins the Kafka Producer and Kafka Consumer. The producer periodically and frequently listens for queries made to the Operational Datastore. When this query occurs, which is executed by the ODS Admin or a health partner, the Producer receives the exact query from the ODS. The producer then publishes this data to a topic, where it is picked up by the Data Transformer inside of the Kafka Cluster. The Data Transformer converts the information it receives into a format that is compatible with the Reporting Database. Meanwhile, the Kafka Consumer has subscribed to a topic to which the Data Transformer will publish this transformed data. Finally, the Consumer reads the transformed data and uploads it to the Reporting Database, concluding the process. This process repeats whenever a new query is made to the Operational Datastore.

The above process is ideal for this particular project because information is constantly being added to and retrieved from the two databases. As a result, a constantly running process that responds instantaneously to any changes made is extremely well-suited for the CDC’s purposes. Additionally, the above diagram shows that the only necessary interaction between our system and an ODS administrator is in starting the Confluent process and querying the database. The ODS administrator is not responsible for scheduling or initiating batch processes, making our solution an improvement to the current SAS implementation in that aspect.

# 

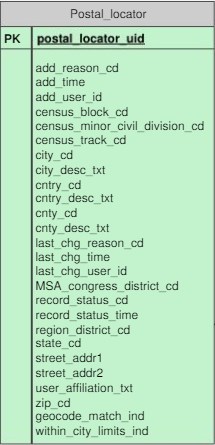
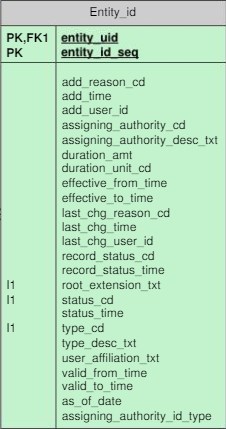
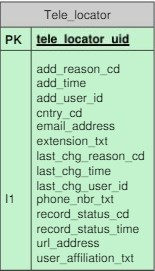
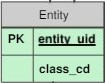
# Data Storage Design

# 

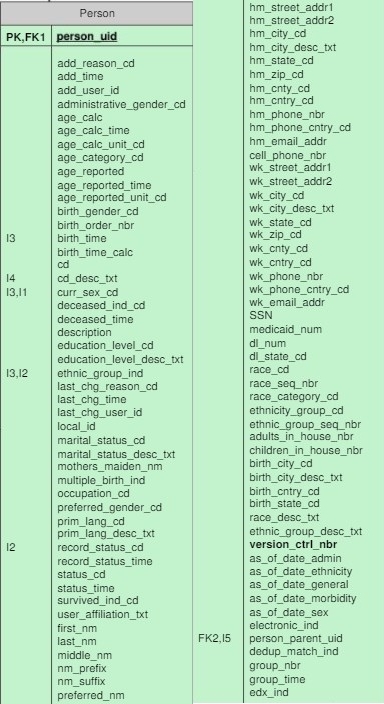
**Figure 3.1** Relationship of main ODS tables

For this project, the focus is on the core Person tables within the ODS. As shown above, a person’s information is stored in six distinct tables, centralized by connecting to the main Person table. Each Entity may have multiple Postal\_locators and Tele\_locators to keep track of current and past residences as well as multiple forms of contact. Each Entity corresponds to one Person and may have multiple names which they refer to themselves with.

Below are the tables with their full contents, which we left out of the above diagram because there are too many entries.



**Figure 3.2** ODS Tables for storing additional supporting person information

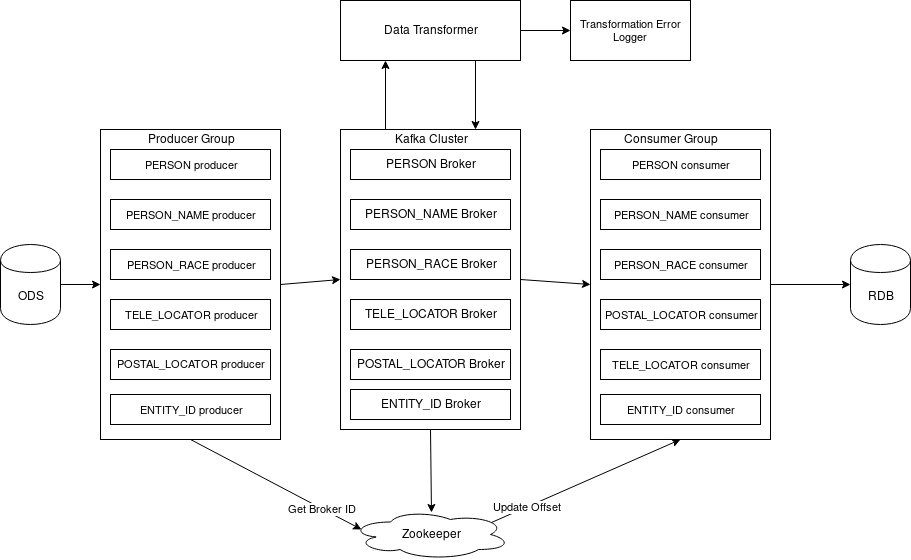


**Figure 3.3** ODS Person table for storing core information about a person

# 

# Component Detailed Design

*Static Design*



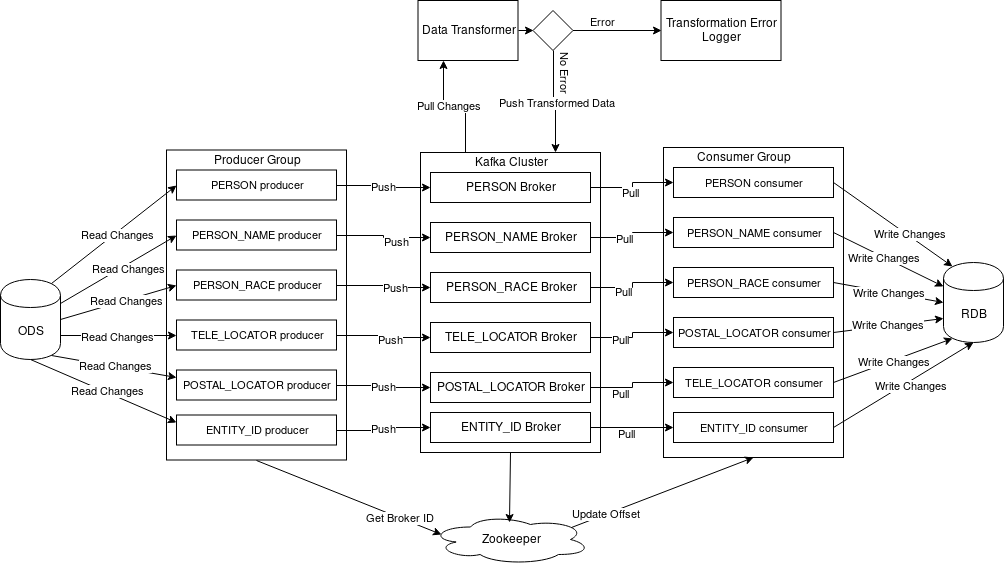
**Figure 4.1** Static Component Diagram

Figure 4.1 shows the static component diagram. Because we are working with Kafka, the only components we will be using are the producers, the Kafka cluster, the consumers, zookeeper, and our operational database and reporting database. Since this is all a backend, black-box process, everything that occurs will be due to users inserting or modifying data in the ODS.

When data is changed in the ODS, the producers corresponding to the changed tables in the ODS will push this change to the Kafka cluster. The cluster maintains the information and logs it, and the consumer pulls this information periodically from the cluster. The consumer than takes the changed data, converts it into the proper format for the RDB, and stores it into the RDB.

Zookeeper is used to organize the connection between the different producers with the Kafka cluster.

*Dynamic Design*



**Figure 4.2** Dynamic Component Diagram

Figure 4.2 outlines the flow of data through each component while the system is running. Each of the producers pulls new changes (insertions or updates) from the ODS, and pushes that data to the corresponding Kafka broker. From here, the Data Transformer pulls all changes and executes our specified SQL transformations to convert the updates into a form legible by the reporting database. Any issues during this process, like the absence of a matching SQL transformation for the data read in by the Transformer, results in the invalid data being sent to the Transformation Error Logger for reporting. Otherwise, the information is rewritten to the Kafka cluster, where it is pulled by one of the corresponding consumers. These consumers write all changes to the reporting database in the correct format.

# UI Design

As mentioned briefly in the introduction, our project deals exclusively with a backend data transformation process and does not interact whatsoever with any users. Although users interact with the Operational Datastore and Reporting Database through partner-facing user interfaces, the Extract, Transform, and Load process occurs independently of this interaction. Figure 2.1 provides context for our project by illustrating several relevant components and their relationships, and our specific task is to implement new features for the highlighted portion, the ETL process. All of the interfacing that occurs with health partners and administrators occurs outside of the illustrated box, through the NBS Entry/Data Exchange and NBS Reporting Module. Therefore, not only is any UI design beyond the scope of our specific project task, but it is also beyond the scope of the data system with which we are working.

1. https://wwwn.cdc.gov/nndss/nedss.html [↑](#footnote-ref-0)