USAID GLOBAL HEALTH SUPPLY CHAIN PROGRAM

Procurement and Supply Management

|  |
| --- |
| LaDATA (Logistic and Demographic Analysis for Targeted Action)  Using big data for targeted actions for the supply of Family Planning commodities  Maintenance Guide  January 2021 |

The USAID Global Health Supply Chain Program-Procurement and Supply Management (GHSC-PSM) project is funded under USAID Contract No. AID-OAA-I-15-0004.  GHSC-PSM connects technical solutions and proven commercial processes to promote efficient and cost-effective health supply chains worldwide. Our goal is to ensure uninterrupted supplies of health commodities to save lives and create a healthier future for all. The project purchases and delivers health commodities, offers comprehensive technical assistance to strengthen national supply chain systems, and provides global supply chain leadership.

GHSC-PSM is implemented by Chemonics International, in collaboration with Arbola Inc., Axios International Inc., IDA Foundation, IBM, IntraHealth International, Kuehne + Nagel Inc., McKinsey & Company, Panagora Group, Population Services International, SGS Nederland B.V., and University Research Co., LLC. To learn more, visit [ghsupplychain.org](http://www.ghsupplychain.org/)

DISCLAIMER:  
The views expressed in this publication do not necessarily reflect the views of the U.S. Agency for International Development or the U.S. government.

Contents

[Contents 1](#_Toc69308805)

[Acronyms 2](#_Toc69308806)

[Purpose 3](#_Toc69308807)

[Introduction 4](#_Toc69308808)

[Overall Architecture 4](#_Toc69308809)

[Application Components 4](#_Toc69308810)

[eLMIS Integration 5](#_Toc69308811)

[DHS, Worldpop Integration 7](#_Toc69308812)

[Server Configuration 7](#_Toc69308813)

[Application Start/Stop Procedure 7](#_Toc69308814)

[Database Backup/Restore Procedure 8](#_Toc69308815)

[Database Design 9](#_Toc69308816)

[Extract, Transform, and Load (ETL) Process 10](#_Toc69308817)

[Dash Back-End HTML 12](#_Toc69308819)

Acronyms

|  |  |
| --- | --- |
| API | Application program interface |
| DB | Database |
| eLMIS | Electronic logistics management information system |
| ETL | Extract, transform, load |
| FP | Family planning |
| HTTP | Hypertext transfer protocol secure |
| LMIS | Logistics management information system |
| ORM | Object-relational mapping |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Purpose

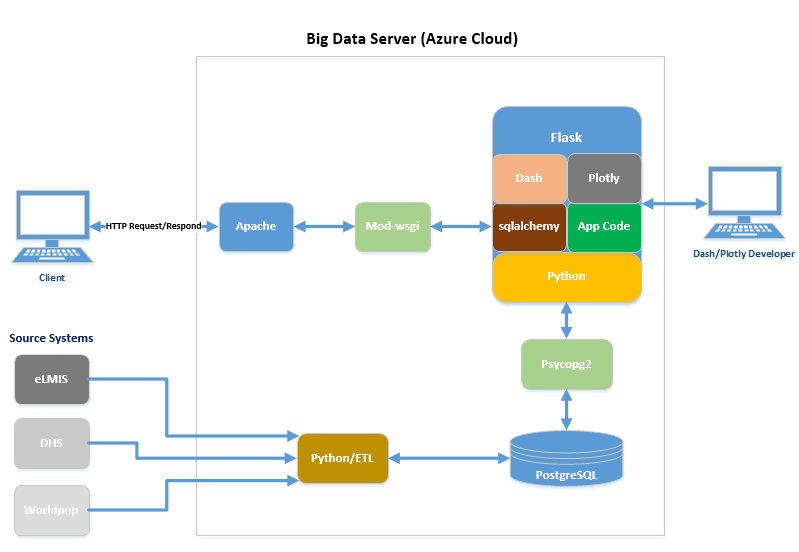
The purpose of this maintenance guide is to explain the information and technology involved in creating the LaDATA (Logistic and Demographic Analysis for Targeted Action, formerly the Big Data) application and how to maintain and update it. It details the overall architecture, the tools used to create the solution, and its integration with the required eLMIS and demographic data sources. This is a resource for maintenance and understanding the components that comprise the tool. For information on using the tool and its purpose, please refer to the accompanying User Guide.

Introduction

This document contains information on how to maintain the LaDATA application. The application involves two separate processes: the extraction, transformation, and load (ETL) and data preparation for front-end visualization.

Overall Architecture

The overall solution architecture for Big Data for Family Planning project contained in the diagram below. The solution incorporates 3 inbound source systems – eLMIS, DHS, and Worldpop. Custom python scripts for extraction, transformation and load (ETL) processes of source data. A central data repository (PostgresSQL) for managing and hosting the processed data. A custom web-based business intelligence application (Dash, Python) for data visualizations and analytics.



Application Components

**Apache -** Web server (open-source) software which allows web applications to serve content on the web.

**mod\_wsgi -** An Apache HTTP Server module that provides an interface for hosting Python based web applications under Apache web server.

**Flask -** Lightweight web application framework written in Python. It provides user with libraries, modules, and tools to help build Web-Applications.

**Flask-SQLAlchemy -** Is an extension of Flask application. It’s a library that facilitates the communication between Python programs and databases. This library is used as an Object Relational Mapper (ORM) tool that translates Python classes to tables on relational databases and automatically converts function calls to SQL statements.

**Application code -** Custom application build on python using the Flask and Dash framework.

**Python -** An object-oriented programming language that provides rapid application development.

**Psycopg2 -** Is the PostgreSQL database driver for the Python programming language.

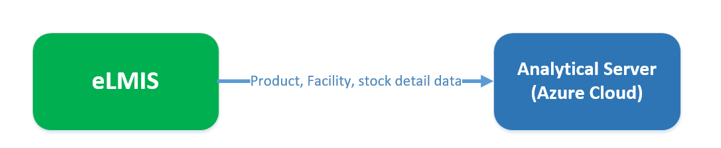
**PostgreSQL -** An open-source object-relational database system.

**Dash -** Dash is a Python framework for building analytical web applications. No JavaScript required. Built on top of Plotly.js, React and Flask, Dash ties modern UI elements (dropdowns, sliders, and graphs) directly to Python code.

# eLMIS Integration

Below is a high-level integration solution.

* File Transfer Protocol: Web Service API
* File Transfer Frequency: Monthly
* Data files:
* Product Master (only the Family Planning (FP) products)
* Facility Master
* Monthly Stock Details (only for FP products)
* Facility Reporting (all facilities)



**Solution Detail**

Below is the design for the integration between eLMIS and Bigdata analytical server using Web Service API’s.

* Build 4 web service API’s on eLMIS end. The API’s can be called from GHSC-PSM analytical server on a scheduled basis (monthly) to retrieve facility, product, stock detail and facility reporting data.
* The file format is pip delimited csv. The file contains a snapshot of one-month data based on 3 parameters (month, year, product\_subgroup) passed in the API call.

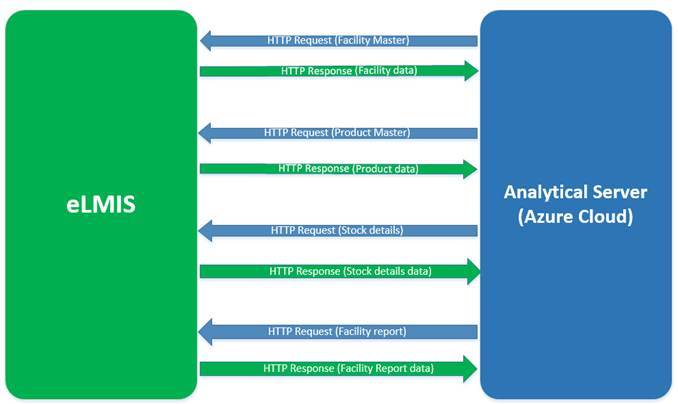
**Web service API**

* GHSC-PSM team will retrieve CSV data using HTTPS GET request.
* GHSC-PSM will trigger a request using provided URL and file parameters.
  + For example, to retrieve a list of all facilities in CSV file format, issue a HTTP GET to the following URL: <https://eLMIS-API-URL/facility.do?CSV>
* The Content-Disposition HTTP header in the response indicates the file name and extension of the extract; "facility.csv" in the example.

**Web service parameters**

The following URL parameters will filter the CSV content.

* For facility master there is no parameter, retrieve all the facilities.
* For product master use parameter: &product\_subgroup=[FP] to retrieve only the list of products for family planning.
* For stock detail use parameters: &year=[year]&month=[month]and product\_subgroup=[FP] to retrieve the list of transactions for family products for a specific month and year
* For facility report use parameters: &year=[year]&month=[month]



# DHS, Worldpop Integration

Currently the demographic data is manually downloaded from the DHS and WorldPop sites. DHS, which stands for Demographic and Health Surveys, was last updated in 2018. WorldPop data is updated on an annual basis and is the source for population and population density. Current version in use is from 2020. WorldPop is also the source for pregnancies data, which was last updated in 2015. DHS is the source of data for geographical coordinates, family planning demand satisfied, family planning unmet need, and use of modern family planning methods. A Python script cleans and combines the two sources into a single table that summarizes the data on the district level. Statistics created for each metric, with the exception of coordinates, are minimum value, maximum value, mean value, median value, range, standard deviation, and count. The script also creates a shapefile for district-level outline.

Server Configuration

Big Data web application server is currently hosted in Azure cloud. Below is the server configuration:

* Server Name: (servername.eastus2.cloudapp.azure.com)
* Application URL: (<https://servername.eastus2.cloudapp.azure.com/dash/>)
* Admin account: psmadmin
* Linux (x86\_64)
* 4 CPU @ 2.6 GHz
* 32 GB RAM
* 128 GB File System

Application Start/Stop Procedure

Below scripts are used to stop/start the database, application server and the mail server.

**PostgreSQL start/stop process:**

Sudo su - postgres

/usr/pgsql-12/bin/pg\_ctl -D /bigdata/postgresql/12/data -l logfile restart

/usr/pgsql-12/bin/pg\_ctl -D /bigdata/postgresql/12/data -l logfile stop

/usr/pgsql-12/bin/pg\_ctl -D /bigdata/postgresql/12/data -l logfile start

DB log path: /bigdata/postgresql/12/data/log

**Apache Server:**

httpd service default configuration

* Default config file: /etc/httpd/conf/httpd.conf
* Default ports: 80 and 443 (SSL)
* Default log files: /var/log/httpd/{access\_log,error\_log}
* systemctl start|stop|status httpd.service
* sudo apachectl restart

**Webapp:**

Path: /var/www/webapp

Virtual environment (venv)

source venv/bin/activate

* Installed components (requirments.txt)
* Wsgi config file (webapp.wsgi)
* Dash source file (webapp.py)

**Mail Server:**

Configuration folder: /etc/mail

Start/Stop Process:

service sendmail stop

service sendmail start

service sendmail restart

/bin/systemctl start sendmail.service

/bin/systemctl stop sendmail.service

/bin/systemctl restart sendmail.service

logs: /var/log

Database Backup/Restore Procedure

Below scripts are used to backup and restore the database.

**DB backup:**

Sudo su - postgres

Backup script path: /bigdata/dbbackup/scripts

./db\_backup\_scheduled.sh

**Current backup schedule:**

00 22 \* \* \* /bigdata/dbbackup/scripts/db\_backup\_scheduled.sh > /tmp/dbbackup.log 2>&1

Retention period: 3 days.

**DB restore:**

createdb -U postgres -T template0 test4

psql -U postgres test4 < bigdata\_07232020.sql

test4 is the new db to be restored.

Database Design

The database (PostgreSQL 12) is hosted in Azure cloud. Below are the Server details:

Host name: servername.eastus2.cloudapp.azure.com

Database: bigdata

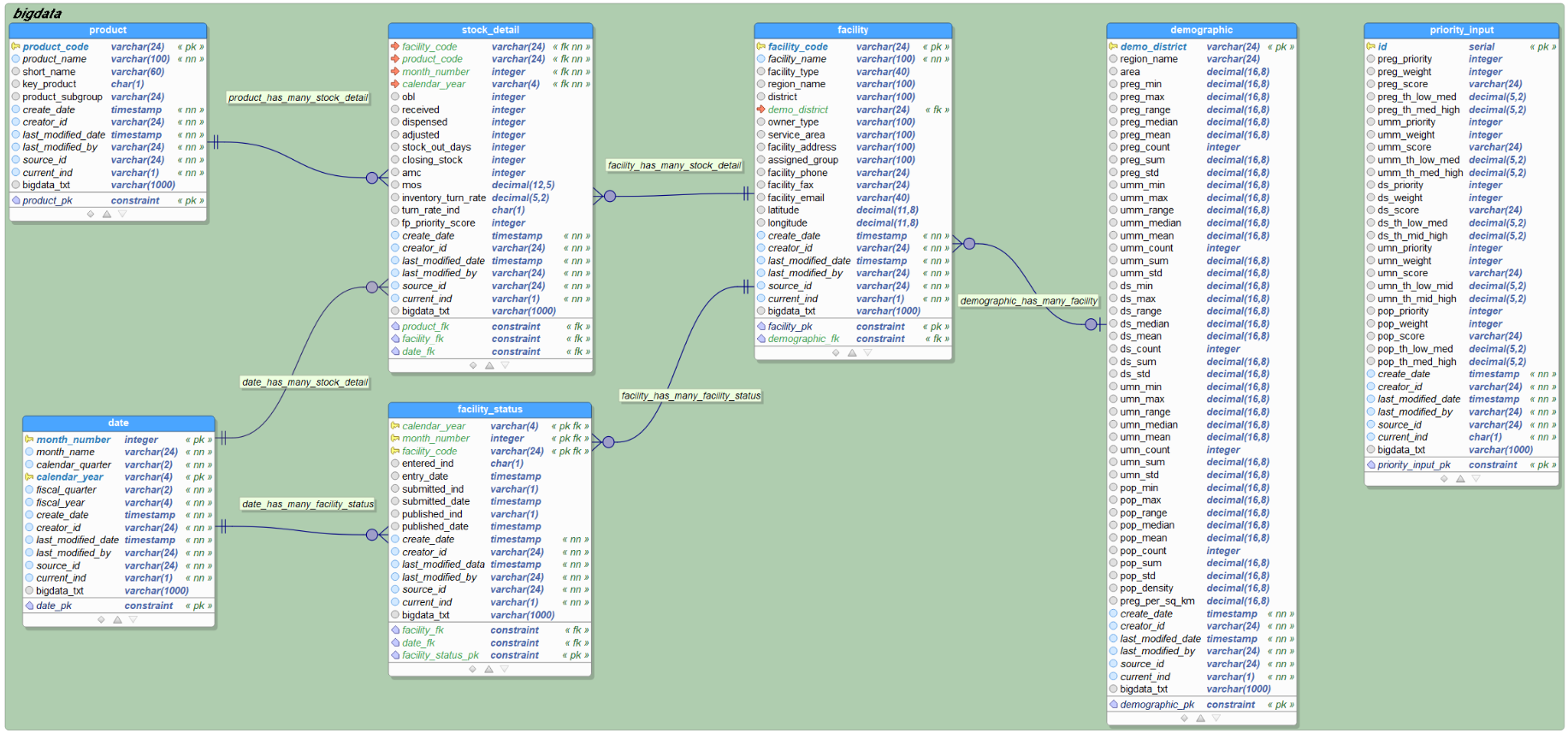
Port: 5432

Schema: bigdata

**Data Model**

There are 7 tables in bigdata schema**.**

* date
* demographic
* facility
* facility\_status
* priority\_input
* product
* stock\_detail



Extract, Transform, and Load (ETL) Process

The ETL process is solutioned on python scripts. The current process runs on a monthly schedule (28th of the month) to extract, transform and load eLMIS data into Bigdata database.

Scripts path: /bigdata/scripts

Logs Path: /bigdata/logs

Source files Path: /bigdata/data/source/elmis

**Scripts:**

automated\_process.py - This script is used to automate the ETL process on a schedule basis. It requires the ‘scheduled’ as an input parameter while calling the script.

Example:

python3 automated\_process.py scheduled

extract\_process\_adhoc.py – This script is used to run the extract process on a manual basis. It requires the ‘adhoc’ as an input parameter while calling the script. It also asks the user to input the desired year and month for the extract.

Example:

python3 extract\_process\_adhoc.py adhoc

Parameter selected= adhoc

Enter desired year ex. 2020 :2020

Enter desired month ex. 01 for Jan :01

User entered year: 2020

User entered month: 01

transformation\_load.py – This script is used for transformation and loading of the source eLMIS data into the PostgreSQL database.

**Cron Job:**

Below cron is schedule to run on a monthly schedule on 28th of the month at 3 am UTC.

0 3 28 \* \* /usr/bin/python3 /bigdata/scripts/python3 automated\_process.py scheduled > /tmp/cron.log 2>&1

# Example of job definition:

# |---------------- minute (0 - 59)

# | .------------- hour (0 - 23)

# | | .---------- day of month (1 - 31)

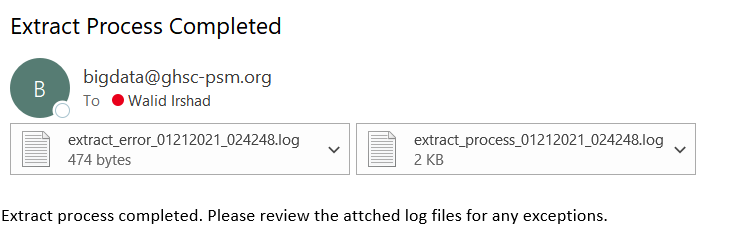
# | | | .------- month (1 - 12) OR jan,feb,mar,apr ...

# | | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat

# | | | | |

**Email Notifications:**

The ETL process generates email notifications when the process is complete. The ETL admin will get an email alerts on each run (sample Below):



Parameter selected = scheduled

system entered year: 2020

system entered month: 12

## Log files path

/bigdata/logs

## Log files name..

extract\_process\_01212021\_024248.log

extract\_error\_01212021\_024248.log

## Data files path

/bigdata/data/source/elmis

## Data files name generated...

product\_master\_122020.csv

facility\_master\_122020.csv

facility\_status\_122020.csv

stock\_detail\_122020.csv

## API URL's generated...

https://sourceURL/product.php?format=CSV&product\_group=FP

https://sourceURL/facility.php?format=CSV

https://sourceURL/facility\_report.php?format=CSV&year=2020&month=12&product\_group=FP

https://sourceURL/stock\_detail.php?format=CSV&year=2020&month=12&product\_group=FP

## Extract process started on : 2021-01-21 14:42:48.763463

Extract process completed for data file: product\_master\_122020.csv

Extract process completed for data file: facility\_master\_122020.csv

Extract process completed for data file: facility\_status\_122020.csv

Extract process completed for data file: stock\_detail\_122020.csv

## Extract process completed on : 2021-01-21 14:42:51.645007

# Dash Back-End HTML

There are three scripts that prepare the data and define the interaction with the front-end Dash application.

**Database Queries:**

The file query.py has SQLAlchemy ORM class definitions for querying facility, product, product, stock detail, priority input, and demographic tables. If the priority input and threshold table is empty, the script will initialize one.

**Table Transformation:**

The file transformation.py is responsible for creating dataframes that can be easily used in visualization. To plot maps against various metrics in Plotly, the shapefile is converted into a GeoJSON. For demographic data, pregnancies per 1,000 people is calculated. The demographic table is then merged with the threshold queries from the database so distribution against threshold can be plotted.

For each district, 12-month inventory turn rate is calculated on the product level in the stock detail. This table is then divided into current year (2020) and historical (2018-2019). The products are identified by their product code in Mali’s LMIS database. The current year data is merged with priority input table to support visualization against turn rate and priority score.

**Webapp:**

The web application is built using Dash with an external CSS stylesheet. The overall design of the front-end application layout is as follows:

Each tab contains the Dash HTML page design. Within each tab, there are interactive components with callbacks setup to update the page based on user selection.

Demographic Parameters:

The data table takes in values from the priority input and threshold table created in transformation.py. The table is editable in only the threshold columns. The histogram and heat map use the data table as an input and plots only selected row. To save the user input in the database, data in both the demographic parameter tab and indicator priorities need to be combined and inserted together. Because global variables are not allowed in Dash to pass data between tabs, the Save and Continue button creates a dataframe based on the current values in the data table and saves it as a JSON on the application level so it can be accessed by other tabs.

Indicator Priorities:

The indicator priorities tab consists of 5 drop-down lists and 5 radio items. Each of them 10 items will modify a hidden JSON item on the tab in their callback functions to record user input. The Submit button will take in the 10 JSON files, along with the threshold file that was created in the demographic parameter tab and saved to the application and push a single insertion to the database.

Dashboard:

Dashboard tab uses the dataframes created in transformation.py for current and historical visualizations. In the current year section, the product drop-down filters the current year inventory turn rate and priority score table and creates 4 visualizations. The first is a scatterplot with priority score on the x-axis and inventory turn rate on the y-axis. The horizontal levels that define the target area are defined based on the quarterly distribution cycle in Mali. Should the distribution cycle change, the target zone should shift accordingly. The second is a data table that lists the values in the scatterplot. This data table is sortable but not editable. The histogram is constructed by region. Each region filters the current year turn rate priority table and has a manually defined HTML hex color codes. The final graph for current year is contains 2 Plotly Choroplethmapbox data. The first filters the current year data with the product selection and plots the inventory turn rate; the second is a grayscale on the same layout that plots districts with no turn rate data.

In the historical section, the historical turn rate is filtered by product and region selection to create a Plotly line chart color-coded by district. The second visualization is only dependent on the product filter and contains 2 traces. The first trace is a heat map of the current priority score by district. Overlaying on top of it is a scattermapbox plotting the inventory turn rate by facility as of December 2019.

The dashboard tab also includes a section plotting just the demographic indicators. This is the same plot as the heatmap in the demographic indicator tab.