Harvard
India Policy Insights project

Architecture and operations

Version/Revision History

Version	Date	Maintainer/Owner	Contact
1.0	10/16/2024	Joaquin Kachinovsky	joaquink@onetree.com

Table of contents

Contents

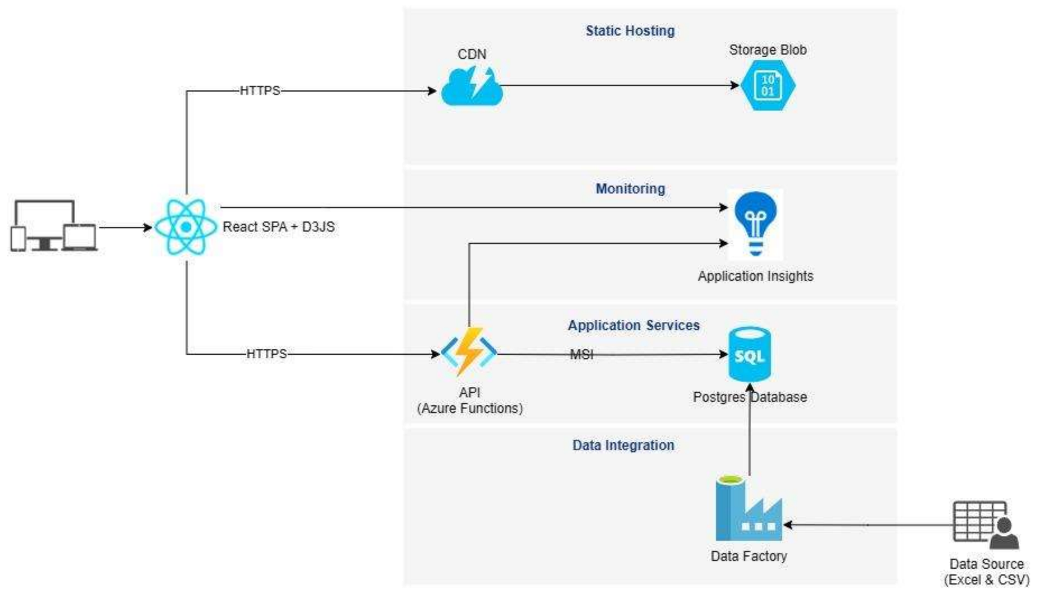
Document Objectives

This documents provides information about the architecture and technologies used to develop the India Policy Insights application, and explanation on how to deploy and run the solution.

Azure Cloud

Hosting the services in the cloud provide several benefits such as scalability, security, and cost savings by paying for just what is used and reducing maintenance costs.
Azure is one of the top cloud providers, and meet the project requirements of having a data center in India and services like Data Factory for ETL purposes and PostgreSQL database hosting.

Below is the architecture of the application hosted in Azure:



The components used are:

- SPA for the frontend, server by an Azure CDN and hosted in a Azure Blob Storage. Blob Storage is a storage solution in the cloud optimized for storing unstructured data such as files, so the files from the build output are stored here, and are served to the user through Azure CDN to leverage the benefits of a content delivery network.
- Azure functions for the backend, also a dockerfile for this component is provided so the functions could be stored in any service that supports docker.
- PostgreSQL database
- Azure Data Factory, used to import data from Excel and .csv files into the PostgreSQL DB. We used the raw data files provided by IPI to allow future compatibility with updated files without requiring any change. Most of the data is in the Excel file except from the villages indicators data, that due to the volume it had to be splitted into

multiple .csv files.

- Application insights to monitor both frontend and backend applications. This service provides real-time analytics about the performance and failures in the platform. The dashboard allow to search for specific requests and deep dive into failures, where it can be seen for example the request that failed, content of the request and which dependencies where part of it such as database calls.

Production environment

Currently in production we have the Url https://indiapolicyinsights.org.in/ as the main entry point, this url is served by the Wordpress instance running in the app service "ipi-wordpress".
This wordpress site has different links to the data explorer at:
https://dataexplorer.indiapolicyinsights.org.in which is the react app hosted in the app service "app-ipi-explorer" that consumes data from the API in the app service named: "ipi-api-prod".
The data explorer also consumes the map data from TileServer in app service: "app-ipi-tiles".

Currently the app services are running at the minimum tier level since there is no "real" users traffic, but the app services can auto-scale in case the traffic increases and more compute power is required.
Also ApplicationInsights is currently tracking all telemetry from the explorer providing performance information and logs of errors in case any is raised.

Different stress testing were conducted with this setup that are provided in the shared folder in the file "Load and stress testing.pdf".

Technologies

Asp.Net core 3.1

Asp.Net core is a cross-platform and open-source framework developed by Microsoft. It is based on a well-known framework (Asp.Net framework) but with several enhancements for improving performance and flexibility.

Reactjs + D3js

React is an open-source JavaScript library used for frontend development, its component-based library lets you build high-quality user-interfaces for web apps, and is one of the most used library currently.
D3.js is also open-source and is used in combination with React to create the data visualizations, is lightweight and works well with large datasets, and provides great flexibility for creating the different visualizations.

PostgreSQL

PostgreSQL is an open-source relational database system, with over 30 years of active development. It provides a good performance for large amounts of data and provides an extensive set of capabilities for this kind of projects that would improve the performance such as table partitioning and columnstore indexing.

Repositories

We have 3 repositories for the application which are:

1. Backend – contains the .Net solution for the backend, dockerfile and scripts to create the database schema
2. Frontend – contains the React.js application
3. Data Factory – it contains the Azure Data Factory configuration including the pipelines to ingest data from the files into the database

How-Tos

Using docker compose

In the backend repository a docker-compose file is provided at /src/docker-compose.yml, the compose file will run the backend API (port 80), PostgreSQL instance and a TileServer instance (port 8080).
Once the services are running, the required tables and data could be restored into the database using the provided dump (specified here) and the provided files in “tiles” folder should be copied to the “tiles” named volume specified in the docker-compose file.

Deploy backend

The backend can be deployed by downloading the publish profile from the Azure Functions resource and import it into the Visual Studio solution to deploy it.
Also, a dockerfile (at /src/Dockerfile) is provided to create a docker image for the application that could be then run in any service that supports docker containers.

Deploy frontend

To deploy the frontend the React application should be built with the npm commands setting accordingly the required variables in packages.json existing commands.
Once the build folder is created it needs to be uploaded to the $web folder of the Azure Blob Storage service and if it is an update is recommended to do a purge of the CDN.

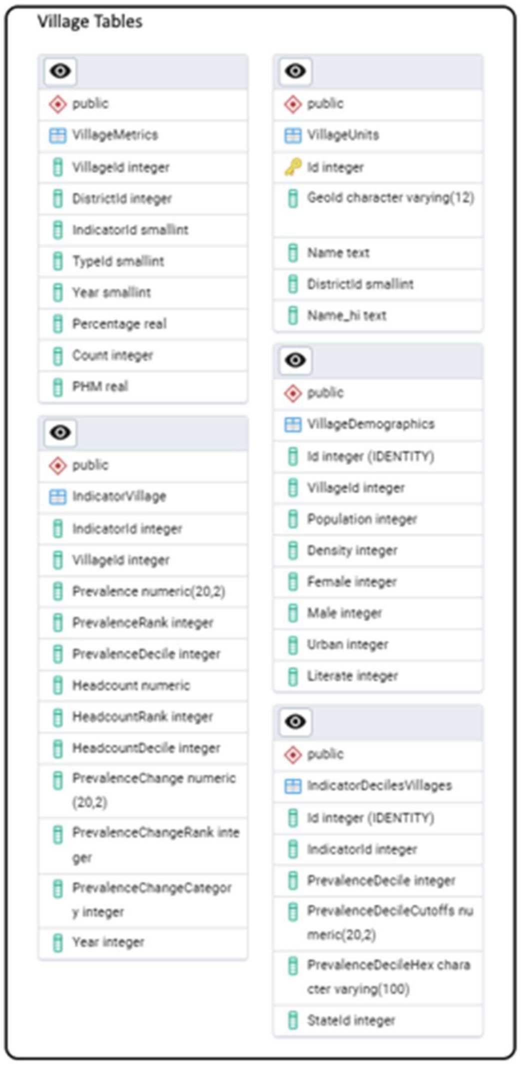
Deploy database

A dump of the current database is also provided in the shared folder, that can be restored to an instance of PostgreSQL using the following command:

pg_restore -Fc --host={server host} --port=5432 --dbname=ipidb --username={username} --clean {path}/IPI_db.dump

In the shared folder an ERM diagram of the database is provided (Database ERM.png), each geography uses a set of tables using the same structure, with some minor variations depending on the data available.

For example for Villages the tables are:


Village Tables

public
VillageMetrics
VillageId integer
DistrictId integer
IndicatorId smallint
TypeId smallint
Year smallint
Percentage real
Count integer
PHM real

public
IndicatorVillage
IndicatorId integer
VillageId integer
Prevalence numeric(20,2)
PrevalenceRank integer
PrevalenceDecile integer
Headcount numeric
HeadcountRank integer
HeadcountDecile integer
PrevalenceChange numeric (20,2)
PrevalenceChangeRank integer
PrevalenceChangeCategory integer
Year integer

public
VillageUnits
Id integer
GeoId character varying(12)
Name text
DistrictId smallint
Name_hi text

public
VillageDemographics
Id integer (IDENTITY)
VillageId integer
Population integer
Density integer
Female integer
Male integer
Urban integer
Literate integer

public
IndicatorDecilesVillages
Id integer (IDENTITY)
IndicatorId integer
PrevalenceDecile integer
PrevalenceDecileCutoffs numeric(20,2)
PrevalenceDecileHex character varying(100)
StateId integer

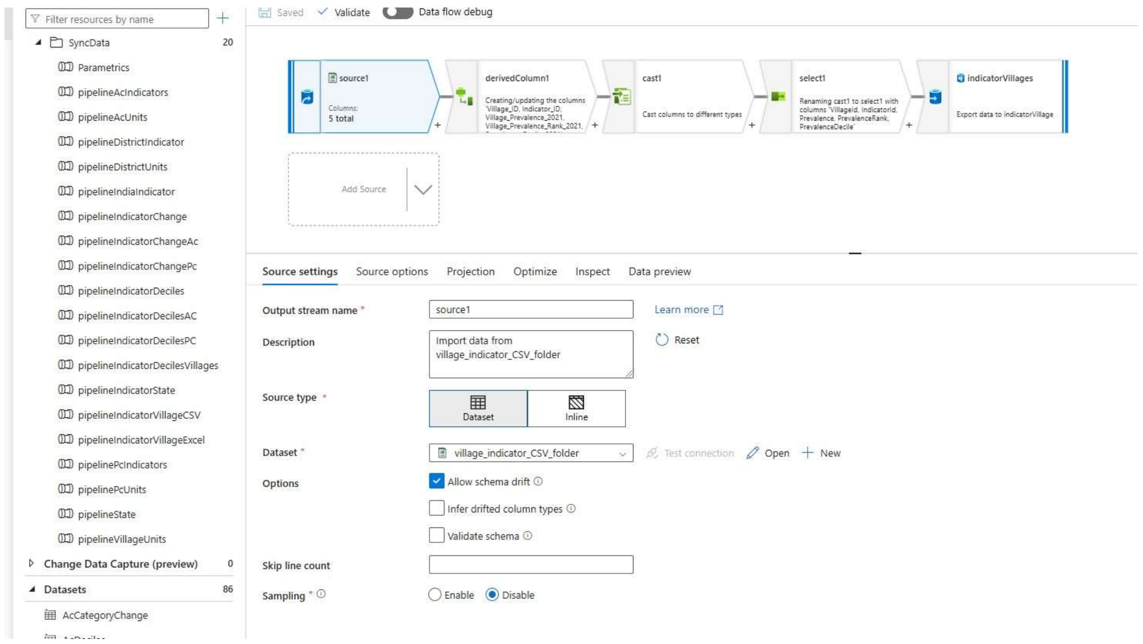
- VillageUnits contain the basic information of each village and to which district they belong to.
- VillageDemographics contain general information about each village.
- IndicatorDecilesVillages stores for each indicator the different deciles based on villages indicators data
- VillageMetrics and IndicatorVillage stores the actual indicator values for each village. These tables are the biggest ones since for each village and each one of the indicators we store values, leaving more than 600 million rows.

Since the data in these tables is read-only (it's only updated by Data Factory) we could use multiple indexes to improve the read performance without affecting the overall performance, allowing a smooth navigation of the data from the explorer.

Data updates

The data factory repository includes all the pipeline configurations to import the data, the new instance of Data Factory would need to be connected to this repository files to import all the configuration.

Once the configuration is imported in Data Factory the files with the new data would need to be uploaded to the corresponding folders and then execute the according pipelines, below is an example of a pipeline:



Filter resources by name

SyncData 20

- Parametrics
- pipelineAcIndicators
- pipelineAcUnits
- pipelineDistrictIndicator
- pipelineDistrictUnits
- pipelineIndiaIndicator
- pipelineIndicatorChange
- pipelineIndicatorChangeAc
- pipelineIndicatorChangePc
- pipelineIndicatorDeciles
- pipelineIndicatorDecilesAC
- pipelineIndicatorDecilesPC
- pipelineIndicatorDecilesVillages
- pipelineIndicatorState
- pipelineIndicatorVillageCSV
- pipelineIndicatorVillageExcel
- pipelinePcIndicators
- pipelinePcUnits
- pipelineState
- pipelineVillageUnits

Change Data Capture (preview) 0

Datasets 86

- AcCategoryChange

Saved ✓ Validate Data flow debug

source1 — Columns: 5 total

derivedColumn1 — Creating/updating the columns 'Village_ID, Indicator_ID, Village_Prevalence_2021, Village_Prevalence_Rank_2021,

cast1 — Cast columns to different types

select1 — Renaming cast1 to select1 with columns 'VillageId, IndicatorId, Prevalence, PrevalenceRank, PrevalenceDecile'

indicatorVillages — Export data to indicatorVillage

Add Source

Source settings | Source options | Projection | Optimize | Inspect | Data preview

Output stream name *	source1
Description	Import data from village_indicator_CSV_folder
Source type *	[✓ Dataset] Inline
Dataset *	village_indicator_CSV_folder
Options	[✓] Allow schema drift; [ ] Infer drifted column types; [ ] Validate schema
Skip line count	
Sampling *	Enable [✓ Disable]

At the left you can see the implemented pipelines , each one to import a different type of data, in most of the cases the pipeline reads from one excel file, processes the data using a Data Flow and inserts the data into the database.
In this case the pipeline in the image is used to import Villages indicators data, since in this case the volume of data was very high the data source was provided in multiple .csv files. Then we leveraged a feature of Azure Data Factory to import data from a folder instead of a file, and each file inside the folder is processed using the same logic defined in the data flow, and then inserted into the appropriate table.

Tiles generation

Harvard provided multiple shapefiles for the different geographies, but this type of file cannot be used on a web environment so we needed to transform them to an appropriate format. Because of the type of data we wanted to display in the maps the use of raster tiles was not a good choice so it was decided to use vector tiles.

To achieve this transformation we first used QGIS open source tool (https://www.qgis.org/) to clean and transform the shapefiles into a geojson format.
Then we used Mapbox tippecanoe (https://github.com/mapbox/tippecanoe) command line tool to generate the different .mbtiles files required by TileServer service to serve the maps.