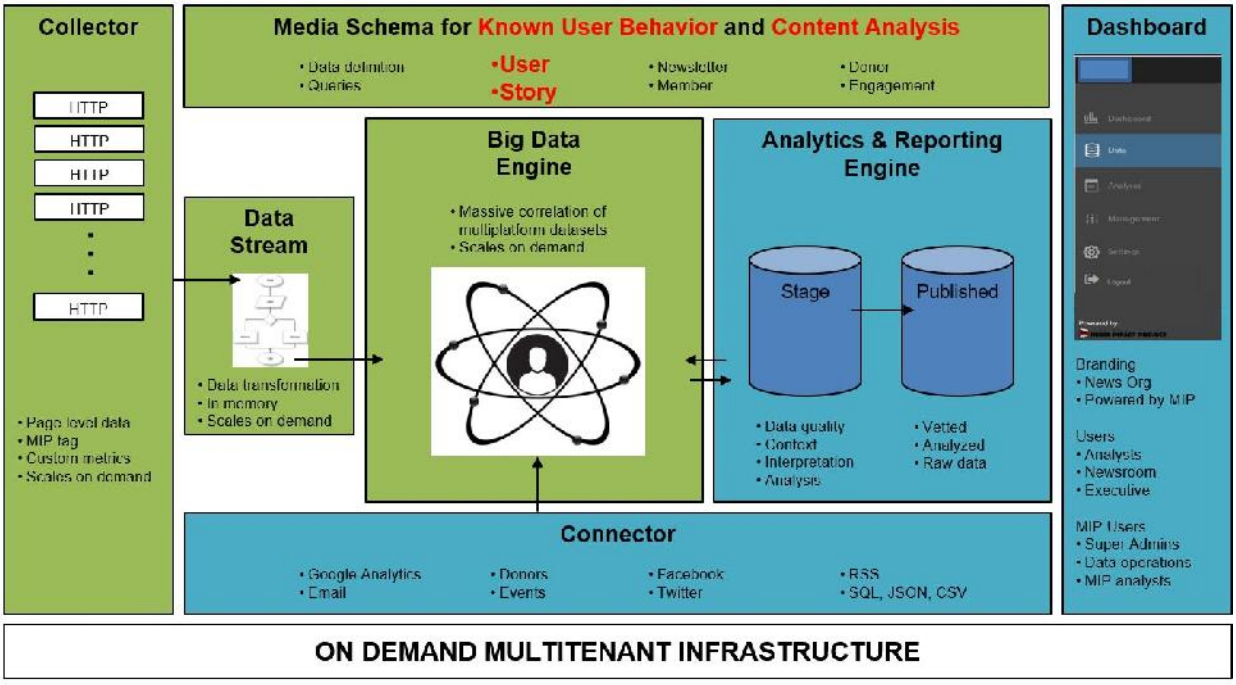
MIP Data Repository & Dashboard Architecture

The Media Impact Project Data Repository brings together data from multiple media products, allowing for assessments of impact not feasible with typical data silos. The repository provides content-centric and audience-centric views of data to understand the impact of stories on individuals.

The Data Repository uses the Google App Engine, which is part of Google Cloud Platform (PaaS or platform as a service), and takes advantage of Google's experience in managing and scaling servers. This allows our developers to focus on higher-level application development, scripting, database schema design, data processing and correlation, etc.

> "*With App Engine, Google handles most of the management of the resources for you. For example, if your application requires more computing resources because traffic to your website increases, Google automatically scales the system to provide those resources. If the system software needs a security update, that's handled for you, too.*"



Google Tag Manager (GTM)
We use Google Tag Manager for sending our customized media-focused metrics and other analytics-related website data asynchronously to our Data Repository Collector application. GTM's built-in version control, debugging, testing, and tag organization are used to ensure the quality of the initial onboarding process and ongoing web data collection.

Data Repository Collector
The Data Repository Collector is a Java application that runs on Google App Engine. This application streams data received from Google Tag Manager directly into Google BigQuery, the analytics data warehouse service. The data repository collector was originally written in Java as it was the original and most stable language available on Google Cloud Platform's services at the time of original development.

The collector application was written by LunaMetrics and made available to MIP in binary form. This application is closed source, but is a simple Java application that accepts the POST and GET requests with the hit data and streams the data into Google BigQuery. We decided to use this collector application for this version of the Data Repository to ensure data validity and quality.

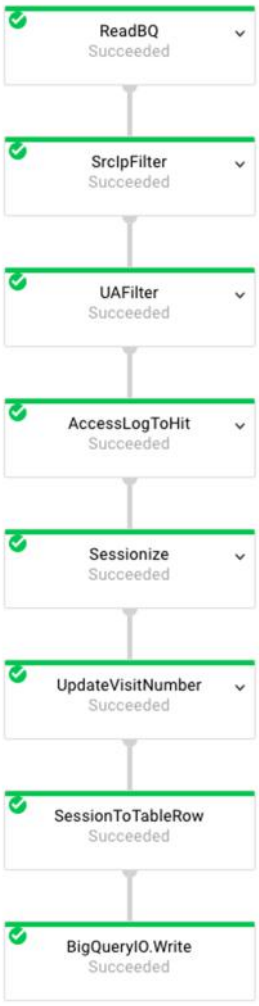
Big Data Engine
The Data Repository Dataflow-Cron application is a lightweight Java application that runs on Google App Engine and kicks off the DataFlow ETL process to transform the raw received data into sessionized hit data comparable to the data available within Google Analytics.

This application was written by LunaMetrics in Java, based on the stability of the Java language on Google Cloud Platform at the time of original development. The application is closed source and provided by LunaMetrics.

Data Processing - From Raw Data to Session Based Data

- The Java-based Dataflow application processes the data on a daily scheduled basis. Dataflow automatically runs a number of processes:

1. **Data Filtering**: Dataflow reads the data from Big Query, then it sends the data through two filters.
    - The first filter allows us to filter out any specific IP addresses, e.g., client employees.
    - The second filter is an overall bot filter to filter out known bot traffic from the results. This filter was developed by LunaMetrics to mimic as much as possible what Google is doing; it's unknown how Google's bots actually work.
2. **Filtered Data to Hit Data**: DataFlow processes the filtered data payload into named paired values based on standard Google Analytics schema.
3. **Processed Hit Data to Sessionized Data:** Dataflow takes the processed hit data and sessionizes the data.
    - The sessionization partitions data by the Google client ID (user), orders it by the time the hits occur, and groups by Google Analytics' session rules, e.g., a session times out after 30 minutes of inactivity.
4. **Update Visit Number within Sessionized Data**: Dataflow updates the sessionized data with the visit number for the session based on the Google client id.
5. **Session to Table Row**: The session is added as a row of data in the table.
6. **Write Table to BigQuery:** The table is written to BigQuery.
7. **Processed Data Within BigQuery:** The final processed data is then available within BigQuery for manual and automated queries.



Data Repository Dashboard
The dashboard or reporting interface application consists of two parts: the front-end for the client and MIP administrators, and the analytics and reporting engine extract-transform-load (ETL) processes for the MIP administrator to use to stage and review data for quality.

Both of the front-end parts of the dashboard are PHP-based applications that run on Google App Engine using the commonly used Laravel framework for modular, agile feature development. The dashboard uses a Google Cloud-based MySQL database.

The dashboard includes automated ETL processes that run queries to extract the necessary data from the data source and enter it into the MySQL database to be available for the dashboard to display. The current ETL processes are written as Python scripts organized to work with Google Cloud Platform. The dashboard runs a weekly scheduled automated process that runs standard queries from the sessionized client data in Google Big Query to extract the story-level data from BigQuery and Google Analytics into MySQL.

BigQuery as part of the ETL processes
The Dashboard executes a set of queries prior to running the standard story-level and user queries. These preliminary queries update MIP user databases which hold identifiers for all users from the time MIP began collecting. After these prepared queries are run, any visitors from the current week who were not already in the databases will be captured and added. Story-level and user queries which extract and calculate the data totals displayed on the Dashboard are then executed following completion of the preliminary queries.

Story URL combining
After examining the story-level reports that were generated directly from the queries, we noticed a number of articles had multiple titles. These stories were all linked to the same unique URL, but each title had its own entry in the report. To improve count accuracy and clean these results, we run an SQL script over the BigQuery report to combine the story data from the same articles. This script combines report entries based on URL, since it is the only constant identifier among the multiple title entries. Upon completion, the output of this script will contain a single field of all titles for a specific article, the unique story URL, and a sum of all story-level data across title entries.

MIP Data Repository Dashboard - Google App Engine Technology Choices

Currently Google App Engine has two types of server environments, standard and flexible. The flexible app engine environments mean that the development of the backend server architecture in that language is still in **beta** version. While some of these language environments are in the release candidate stage and will soon graduate into the standard environment, we have based our applications on standard environment languages in order to fully leverage the stability and documentation of the existing standard environment languages.

Language	Standard Environment	Flexible Environment
Python	Python 2.7	Python 2.7/3.4
Java	Java 7	Java 8
PHP	PHP	
Go	Go	Go
Node		Node.js
Ruby		Ruby
More		Custom Runtimes

See more here: https://cloud.google.com/appengine/docs

Dashboard website technology choices

Below is our reasoning for choosing Java, Python and PHP, and not choosing Node, Ruby, or Go.

1. Java has stable Google Cloud Platform support. We can use it as for complete MVC server-side rendering for html, or just as a RESTful Web API service. Although Java is older and not as developer-friendly, both our developers and LunaMetrics have experience with Java.

2. Python is like Java in that it has stable Google Cloud Platform support, can use web2py framework to develop websites, and support RESTful Web API. Both of our developers have experience with Python. However, Python is not traditionally used as a website language, but as scripting language for server-side tasks.

3. Ruby and Node are popular languages right now and both of our developers have experience with them. However, both are using the beta version BigQuery API client and they only can be used in in the Google App Engine beta flexible environment.

4. Go has been launched on Google Cloud Platform for a long time, but is still a very niche development language.

5. PHP is the most popular scripting website language. Both of our developers have experience with PHP development. It has stable Google Cloud Platform support on Google App

Engine in the standard environment. PHP development is at a lower cost compared to other languages. While the BigQuery client API 2.0 is a release candidate, PHP was our ultimate choice for developing the dashboard website.

BigQuery and MYSQL

In deciding on a database for the dashboard, we evaluated using both BigQuery directly and MySQL. MySQL outperformed BigQuery in our tests for website use. In using MySQL to store the dashboard report data, it reduces the response times. By modularizing the BigQuery and other ETL queries, we can automate and run these processes separately from the dashboard core. This increases website stability and performance.

MVC Server-Side Rendering HTML vs. Single Page Application

1. Server-side rendering HTML operates as traditional website MVC website architecture using server-side technology like Java Ruby, PHP, etc. to render the HTML. Both of our core developers are experienced with this type of website architecture.

2. Single-page application (SPA) uses Ajax or WebSockets to get data from the web server. The server-side only needs to offer pure data API.

Based on our evaluation of SPA on Google App Engine below, we decided to go with a traditional MVC server-side technology.

While we ultimately did not choose to develop an SPA, below are the advantages and challenges with the SPA model:

Disadvantages

1. Search engine optimization for SPA sites is not straightforward.
2. Google Analytics requires more work to setup and accurately collect details as SPA does not treat pages the way GA sees them by default.
3. Development is more complex in Google App Engine.
4. There is a higher cost with hiring developers.

Advantages

1. We can easily use/reuse code in mobile development. Like use cordova framework to build IOS and Android APP.
2. Thin Server Architecture. Moved complexity UI display logical from the server to the client.

Further Challenge of SPA within Google App Engine

The original App Engine architecture is based on a single front-end instance with optional backend instances. With SPA there are two front-ends. One is Java Restful Web API; the other, the SPA web application.

Normally SPA architecture methodology would not be recommended to host static files in Java, even though it can. The SPA methodology uses Nginx/Apache to host static files and send the request of dynamic pages to back Tomcat. While we could use Google Cloud Storage for hosting static files, this adds another layer of complexity to our website. See more here https://cloud.google.com/storage/docs/hosting-static-website

Choice of PHP Frameworks

The number of PHP web frameworks allows for fast development of features. Based on SitePoint's annual 2015 survey, the top PHP frameworks used at work and on personal projects were: Laravel, Symfony, Nette, and CodeIngiter, with Laravel being more the most popular by far.
https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/

1. Laravel is an open source framework "intended for the development of web applications following the model–view–controller (MVC) architectural pattern. Some of the features of Laravel are a modular packaging system with a dedicated dependency manager, different ways for accessing relational databases, utilities that aid in application deployment and maintenance"

(See https://en.wikipedia.org/wiki/Laravel)

While Laravel is a newer framework (2011), it built on lessons learned from older framework successful PHP application frameworks. It even leverages some of the Symphony codebase. The core developer took the time to develop quality professional level documentation and has a monthly fee based laravel training video catalog and service with 700+ screencasts. (see https://laracasts.com) It has a strong community following and is growing in popularity.

Laravel has an SPA compatible version (Lumen) that leverages the same core software. If we decided to switch to SPA, we could use Lumen + React, etc.

2. Symphony is an MVC-based PHP framework and was the previously most popular PHP framework. It is modular by design, with knowledgeable community following. However, Symphony structure design is complex and requires much initial configuration and not desirable for bringing new developers up to speed quickly.

3. Nette is a modular PHP framework in active development with many components, while there is some community support and popularity it is less than half as popular as the leading framework (Laravel). However, the original core developer is no longer involved and quality of the documentation leaves much to be desired.

4. CodeIngiter was a popular MVC based PHP framework with backing of a popular CRM developer (Expression Engine.). However, the original development company has moved on,

and original code branch is no longer being actively maintained. Expression Engine is being rewritten from ground up to not include any CodeIgniter. While a new company is trying to push it forward, it does not have much community support.

CSS library: Bootstrap vs Foundation

1. Bootstrap has better support of support IE 8.0
2. However, Bootstrap 4.0 current in alpha stage.
3. Foundation 6 is stable release quality code. There are many new features in this version.

We chose Foundation 6 as it is more stable than Bootstrap 4, and more powerful than Bootstrap 3.