Toolbox Analytics – Level 0 Documentation

**Toolbox Analytics – Level 0 Documentation**

June 2019

Document Prepared by:

Ryan Colwell

Table of Contents

[Toolbox Analytics – Level 0 Documentation 1](#_Toc12517832)

[Document Purpose 2](#_Toc12517833)

[Level Zero Framework Purpose 2](#_Toc12517834)

[Account Setups 3](#_Toc12517835)

[AWS 3](#_Toc12517836)

[GDM (ToolBox Analytics) 3](#_Toc12517837)

[Toolbox Analytics Application 3](#_Toc12517838)

[Local Environment Setup 3](#_Toc12517839)

[Python 3](#_Toc12517840)

[Repository Setup & Cloning 3](#_Toc12517841)

[Toolbox Analytics SpineDS Library Installation 3](#_Toc12517842)

[Toolbox Analytics Level Zero Framework Repository 4](#_Toc12517843)

[Toolbox Analytics Level Zero Documentation Repository 4](#_Toc12517844)

[IDE Setup 4](#_Toc12517845)

[Development Process 5](#_Toc12517846)

[File Evaluation 6](#_Toc12517847)

[Code Configuration 8](#_Toc12517848)

[GE Job Creation 10](#_Toc12517849)

[Project Configuration 10](#_Toc12517850)

[Activity Resources 12](#_Toc12517851)

[Script Sources 12](#_Toc12517852)

[Executing a Project 12](#_Toc12517853)

[Data Feed Documentation 12](#_Toc12517854)

[Content Guidelines: 12](#_Toc12517855)

[User-facing Docs: 13](#_Toc12517856)

[Development Docs: 14](#_Toc12517857)

[Configuration Instructions: 14](#_Toc12517858)

# Document Purpose

This document describes the configuration process of the Toolbox Analytics Level Zero Data Transformation framework. The intended audience is the technical team that will be configuring new and existing feeds within the framework.

# Level Zero Framework Purpose

The Level Zero Framework is used to transform PeopleCloud 3rd Party data feeds into the a workable format for Data Science analysis and additional processing. The outputs of the Level Zero framework are used by the PeopleCloud Data Science team as well as Data Scientists and Analysts that use the Toolbox Analytics application. Analysis performed by these teams varies per client, but is usually focused on better understanding potential and existing customers of the clients to better inform their marketing strategies.

# Account Setups

The following accounts need to be setup to start the Level Zero Transformation configuration process:

## AWS

An AWS account needs to be created to be able to access the code library and machinery needed to run the code.

1. Create an AWS account with the required roles
   1. Email Chaitanya Bijawe for AWS account creation.
      1. AWS Account Id: 5799-2234-9979
      2. Account must have attached policies via role or groups:
         1. AWSCodeCommitFullAccess
         2. AmazonEC2FullAccess
         3. AmazonS3FullAccess
2. Install Git
   1. Download and install Git from <https://git-scm.com/downloads>
3. Create a SSH Key for Code Commit
   1. Follow “Step 3” in here: <https://docs.aws.amazon.com/codecommit/latest/userguide/setting-up-ssh-unixes.html>

## GDM (ToolBox Analytics)

Toolbox Analytics hosts their own version of the Publicis Global Data Management (GDM) tool, which is used to execute Level Zero Transformation processes.

1. Create a GDM Account for the Toolbox Analytics GDM Tool
   1. Toolbox Analytics GDM URL: <https://gdm-analytics-us-east.ppc.publicismedia.com/>
   2. Email Chaitanya Bijawe to have your account created.

## Toolbox Analytics Application

The ability to create EC2 instances will be required during the development process. This can be managed through the Toolbox Analytics Application.

1. To request an account, go to <https://runads.atlassian.net/servicedesk/customer/portal/3/group/77/create/282>
2. Complete the name and agency inputs
3. Input “SPINE” as the client
4. Input “PeopleCloud Dev” as the project

# Local Environment Setup

To setup your local environment you will need to Clone the Level Zero repository to your local machine.

## Python

The Level Zero Framework uses Python 3.6. Download can be found at: <https://www.python.org/downloads/release/python-368/>

## Repository Setup & Cloning

### Toolbox Analytics SpineDS Library Installation

The SpineDS Library contains a number of tools that the Level Zero framework depends on

**Repository Name:** pcs-spineds-lib

**SSH URL:**ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/pcs-spineds-lib

**Terminal Command(s):**

git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/pcs-spineds-lib

cd pcs-spineds-lib

git checkout master

sudo pip3 install .

### Toolbox Analytics Level Zero Framework Repository

**Repository Name:** pcs-spineds-transform

**SSH URL:**ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/pcs-spineds-transform

**Terminal Command:** git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/pcs-spineds-transform

### Toolbox Analytics Level Zero Documentation Repository

**Repository Name:** pcs-analytics-doc

**SSH URL:**ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/pcs-analytics-doc

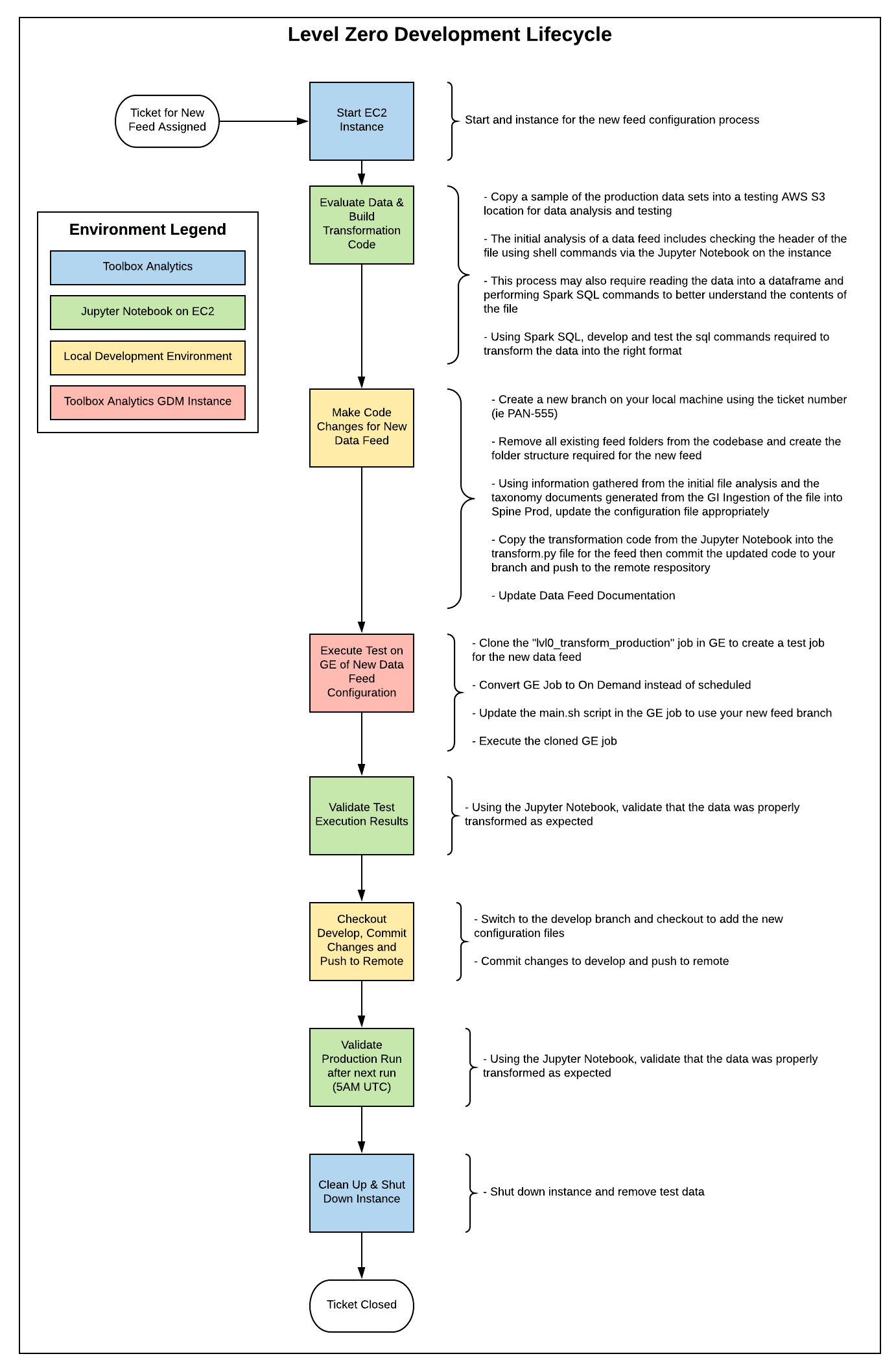
**Terminal Command:** git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/pcs-analytics-doc

## IDE Setup

The preferred IDE for the Level Zero Transformation code is PyCharm. The Community (free) version of PyCharm can be downloaded here: <https://www.jetbrains.com/pycharm/download/>

# Development Process

The following is a high level overview of the Level Zero Framework Development Lifecycle:



## **File Evaluation**

As part of the file evaluation process you must determine the table type. There are four main table types we receive from vendors/partners: IDMap, segment, event, and taxonomy tables. We follow certain conventions on how we transform/store the data based on the table type.

**IDMap Tables:**

IDMap tables provide a look up between a vendor's ID space and our own. The tables typically have just two columns. Cadence for these tables typically matches the associated segment or event table. These tables always come via our matching partner Liveramp rather than directly from a third party data vendor.

Sample Table (source: Dun & Bradstreet IDMap):

|  |  |
| --- | --- |
| **idl** | **source\_id** |
| XY6193UzsKiYlHkVApmUwDeGB5YeuTBofgBeIPTtEyxDsMNRQ | 9abc825b98e11201363d89de21dd0fbb |
| XY6193BbV1LIOk1kUyIH864Or8FPWaKZFW-lPqdVpKxmOw7SM | 1e7924e4372a356f8cd6a482bdd4ac49 |

Level 0 conventions:

* All columns names are lowercase and spaces are replaced by \_ characters.
* All tables are converted to parquet regardless of input format.
  + All tables are repartitioned to ~128MB part files.
* All tables are partitioned by the date format specified in ingestion (Spine Prod S3).
* Assure the idl column (i.e. Liveramp ID, aka our ID space) is renamed to idl.
  + Sometimes data comes to us without a header or the column is named 'device\_id' or 'customer\_link'
* Leave the source ID column named whatever it comes in with.
  + If the source's ID column does not have a header, rename to 'source\_id'

**Segment Tables:**

Segment tables provide a snapshot view on a records' attributes. For example in the US, Experian provides us with demographic information that is refreshed quarterly. These tables can have lots of columns/attributes but are typically proportional to the given market's population. These tables are typically delivered on a monthly or quarterly cadence.

Sample Table (source: Dun & Bradstreet Segment Table, subset of attributes):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **source\_id** | **job\_function** | **sub\_job\_function** | **industry** | **sub\_industry** |
| 31c1bea9615f4a2866c62d61e5dea15e | General Management | Principal | Finance and Insurance | Insurance |
| 9f286039580454693fa1d56b15abb8ef | Finance | Admin | Retail Trade | Vehicle and Parts Dealers |

Level 0 conventions:

* All columns names are lowercase and spaces are replaced by \_ characters.
* All tables are converted to parquet regardless of input format.
  + All tables are repartitioned to ~128MB part files.
* All tables are partitioned by the date format specified in ingestion (Spine Prod S3).
* If non-human readable values in columns, apply associated taxonomy to make it human readable.

**Event Tables:**

Event tables have each row as a record of an event or events occurring. These tables have an ID for a user, a timestamp, and some data about an event occurring. For example, a record might represent a person visiting a website at a given point in time. These tables typically come to us daily and are very long (large number of rows) as a user can have multiple events/records per day.

Sample Table (source: ShareThis Event table, subset of columns):

|  |  |  |  |
| --- | --- | --- | --- |
| **estid** | **standardTimestamp** | **url\_metadata\_canonical\_url** | **mappedEvent** |
| uBt4Hlu38dcAAHKE480pAg== | 2019-06-10T14:18:04.000Z | https://www.gymshark.com/products/gymshark-emp... | pview |
| ZGAMJFzOwTMAAAARc3BmAw== | 2019-06-10T20:40:13.000Z | https://saramoulton.com/books | pview |
| ZGAVMFz+0xQAAAASdK7eAw== | 2019-06-10T22:00:52.000Z | https://thenewstack.io/ibm-open-sources-razee-... | pview |

Level 0 conventions:

* All columns names are lowercase and spaces are replaced by \_ characters.
* All tables are converted to parquet regardless of input format.
  + All tables are repartitioned to ~128MB part files.
* All tables are partitioned by the date format specified in ingestion (Spine Prod S3).

**Taxonomy Tables:**

Taxonomy tables provide a look up between attribute IDs in event or segment tables and human readable values. See sample table for example. Taxonomy tables are typically static and only received once.

Sample Table (source: comScore Census Event IAB Taxonomy):

|  |  |  |
| --- | --- | --- |
| **iab\_category\_id** | **category** | **subcategory** |
| 118801 | Hobbies & Interests | Art/Technology |
| 117702 | Health & Fitness | Allergies |
| 120004 | Law, Gov't & Politics | U.S. Government Resources |

Level 0 conventions:

* All columns names are lowercase and spaces are replaced by \_ characters.
* All tables are converted to parquet regardless of input format.
  + All tables are repartitioned to ~128MB part files.
* All tables are partitioned by the date format specified in ingestion (Spine Prod S3).

## **Code Configuration**

Each table in the level 0 framework has two files that need to be populated for the framework to pick it up and recognize it as a new source: a configuration file (config.json) that contains metadata about the incoming table from the vendor and a transformation file (transform.py) that contains the PySpark code to do the transformation. As a reminder the source code follows the following structure:

* scripts/
  + vendor\_<vendor\_name>/
    - <data feed name>/
      * <table name>/ - usually idmap, segment, or event
        + config.json
        + transform.py

Sample config.json (source: Dun & Bradstreet IDMap):

Note: comments added for documentation purposes and cannot exist in a valid JSON object.

{

    "vendor" : "dnb", # vendor name, **required**

    "data\_feed" : "audience", # feed name, **required**

    "table\_name" : "idmap", # table name, **required**

    "script\_name" : "transform.sql", # leave as 'transform.sql'

    "sort\_cols": [], # array of column names to sort by, usually leave as empty array

    "part\_cols": [], # array of column names to partition by, usually leave as empty array

    "cadence" : "daily", # cadence of feed, optional

    "lookback" : 2, # level 0 look back, optional

    "file\_type" : "csv", # type of input file, options are: csv, parquet, json, and text, **required**

    "delimiter" : "\u00FE", # delimiter/column separator in input file, **required**

    "suffix" : "", # used for files that have hourly partition in input file, **required**

    "header" : false, # boolean for header, **required**

    "schema" : null, # optionally provide schema, optional

    "feeds" : {

        "us-east-1" : [

            {

                "src" : "gdm-gi-ingestion-us-east-1-prod/service/liveramp\_dun\_bradstreet\_us/market/mapping/",

                "src\_part\_type": "%Y/%m/%d",

                "dst" : "vendor-dnb/audience/idmap/",

                "market" : "core-data-us",

                "partitions": 217

            }

        ]

    }

}

**Feeds:**

Each table will have at least one region/feed. Sometimes we get the same table in different markets and multiple feeds will need to be configured. All data for North America go under the key 'us-east-1' (the AWS region where the data exists). All EMEA feeds will go under 'eu-west-1'.

Feed Key/Values:

* src - the s3 source (sans protocol, e.g. s3://) where the input data lives.
  + North American data lives in the s3://gdm-gi-ingestion-us-east-1-prod/service/ bucket.
  + EMEA data lives in the s3://gdm-gi-ingestion-eu-west-1-prod/service/ bucket.
* src\_part\_type - the source partition type. Typically data lives in each S3 prefix but is partitioned by the date received.
  + e.g. s3://gdm-gi-ingestion-us-east-1-prod/service/<vendor/.../2019/06/10/files.csv.bz
  + Sometimes files are partitioned by hour. This is when we need to populate the 'suffix' key in the config.json. In the example below, the suffix would become '00'
    - e.g. s3://gdm-gi-ingestion-us-east-1-prod/service/<vendor/.../2019/06/10/**00**/files.csv.bz
* dst - the destination S3 prefix path. Does not include the bucket name. This path matches the tree format of the level 0 files.
  + vendor-<vendor\_name>/<data\_feed\_name>/<table\_name>/
* market - the destination S3 bucket name.
* partitions - the number of partitions or parquet part files created by the level 0 process. We aim to have 128MB part files for optimization purposes.
  + For example: if a snappy compressed CSV is 50 GB, we want ≈391 partitions because 50GB /128MB ≈ 390.6
  + Python logic: import math; math.ceil(50e9/128e6)

Sample transform.py (source: Dun & Bradstreet IDMap):

def spark\_df\_transformation(spark, df):

    df.createOrReplaceTempView("idmap")

    df = spark.sql("""

    SELECT

        \_c0 as idl,

        \_c1 as source\_id

    FROM

        idmap

    """)

    return df

The transform.py file is very simple. At a minimum it must contains a single Python function called spark\_df\_transformation that takes a SparkSession and Pyspark DataFrame as positional arguments. If no column renaming or transformations need to be done, the function can simply return the data frame object it is passed. However, if a transformation needs to be done, the logic must be done inside the body of the function. The above example registers a temp table to rename the null column names in Spark SQL, but this transformation could be done a number of different ways.

Additional example transformations:

Applying logic to transform nonstandard time value to a UTC timestamp:

Source: vendor\_comscore/web\_panel/traffic/transform.py

 def spark\_df\_transformation(spark, df):

    df.createOrReplaceTempView("traffic")

    df = spark.sql("""

    SELECT

        machine\_id,

        url\_idc,

        person\_id,

        time\_id,

        domain\_name,

        url\_host,

        url\_dir,

        url\_page,

        url\_refer\_domain,

        url\_refer\_host,

        url\_refer\_dir,

        url\_refer\_page,

        mimetype,

        http\_rc,

        keywords,

        html\_title,

        pattern\_id,

        from\_unixtime(CAST(ss2k AS integer) + 946598400) AS timestamp

    FROM

        traffic

    """)

    return df

Applying a schema to a data source that did not have one, then applying a transformation:

Source: vendor\_retargetly/audiences/idmap/transform.py

import pyspark.sql.functions as F

import pyspark.sql.types as T

def spark\_df\_transformation(spark, df):

    idmap\_schema = T.StructType([

        T.StructField("rtgtly\_uid", T.StringType(), nullable=False),

        T.StructField("android\_mapping", T.StringType(), nullable=True),

        T.StructField("ios\_mapping", T.StringType(), nullable=True)

    ])

    df = spark.createDataFrame(

        df.rdd,

        schema=idmap\_schema

    )

    df = df.select(

        "rtgtly\_uid",

        F.split("android\_mapping", ",").alias("android\_mapping"),

        F.split("ios\_mapping", ",").alias("ios\_mapping")

    )

    return df

## GE Job Creation

For creating jobs in GE, there are 3 main components to know - Project Configuration, Activity Resources and Script Sources. For our level-0 transformation work, we work with the prior 2 components to set up jobs.

### **Project Configuration**

The first step in working with the Global Execution component of GDM is to configure a project. A user can log into the GE Project Configuration page by clicking:

1. on large Execution icon on the Home page
2. on the Project Configurations option in Execution in the top navigation bar

On the Project Configuration page, you will see a dropbox on the left to select any existing of the projects and you may land on one of the existing projects. On the right side of the page you will see additional parameters for configuring your project under Project Settings. At the center of the page you will see a circular icon similar to a node in a graph called activity. On clicking the activity node, you will see parameters for configuring the activity under Activity Settings. You can think of each activity as a job to execute a script and each activity has it's execution resource to run the job. Important parameters to configure:

1. **Project Name:** Name of the project, lvl0\_transform\_production in the above example.
2. **Scheduled/On-Demand:** If the activity has to be run daily, the scheduled option has to be checked. If the project has to be run on demand, the On-Demand option must be checked. The Start Hour of the activity can be set as desired by the user.
3. **Execution Resource:** Resource to execute the activity's job on. GE supports AWS EC2, Lambda, Glue (not limited to) as execution resources. In the example, the execution resource is spineds-0.6.2-gdm-3TB, which is an AMI for an EC2 instance pre-configured in the Execution Resources page (more on setting up Execution Resources below).
4. **Wait for Preceding:** Project Run, if the project should wait for the preceding day's project run to finish running. Activity Run, if the the project should wait for the preceding day's activity to finish, or Do Not Wait if waiting is not required.
5. **Execution Scripts:** An execution script is a script for the job associated to an activity. For most of our work, it is a shell script to run our level-0 transformation Python process. An alternative to writing an inline script is that an execution script can also be imported from GitHub. A GitHub repo must be setup in the Script Sources for the latter option.

The screenshot above is the Project Configuration for our level-0 transformation process. The execution script is as follows:

# Pull the latest library code which has the level 0 transformation code.

cd ~/pcs-spineds-lib

git checkout develop

git pull

# Reinstall the library

sudo python3 -m pip uninstall spineds-lib --y

sudo python3 -m pip install -e .

# In home directory, clone pcs-spineds-transform repo from CodeCommit

cd ~/

git clone <https://git-codecommit.us-east-2.amazonaws.com/v1/repos/pcs-spineds-transform>

# Run transformation code from repo directory

cd ~/pcs-spineds-transform/src/

python3 main.py

The project is scheduled to run daily and at each project run an EC2 Instance with **spineds-0.6.2-gdm-3TB** AMI is instantiated to run the above script.

Additionally, for running transformation jobs which are not configured in the pcs-spineds-transform repo in the develop branch, a separate branch can created with the required configs and can be cloned to run using that particular branch. We have configured the sources for UK in another branch named **lvl0\_transform**. The script for running the UK Data transformation process is the same except the line for cloning the pcs-spiends-transform is changed for cloning from the **lvl0\_uk** branch, given as

git clone <https://git-codecommit.us-east-2.amazonaws.com/v1/repos/pcs-spineds-transform>\

    --single-branch --branch lvl0\_uk

We also use this methodology of creating separate branches for testing configurations for new data sources. Once the data source is independently working fine, we merge the configs to the develop branch, which is already scheduled to run daily.

### **Activity Resources**

The Activity Resources page can be reached by clicking on the Execution option in the nav-bar, then clicking on the Activity Resources option in the drop-down menu. You will see a list of already existing resources. An example of an activity resource can be seen in the image below.

The parameters for setting up an activity resource is as follows:

**Source:** Select one from AWS EC2, Google BigQuery, AWS Glue, etc. We use EC2 Instances for for our level-0 transformation GE jobs.

**Resource Name:**Name of resource.

**Resource Description:** Description of resource.

**AMI ID or Name:** ID or Name of the AMI to create an instance of. If there is a mismatch in the name, a default AMI will be used.

**EC2 Size:** Instance type of EC2 Instance.

**Resource Timeout:** Maximum number of hours that the resource can run for one project run.

For our level-0 transformation work, we created an AMI with Python and all other required packages and dependencies. As a reminder, the activity resource can be selected in the Execution Resource dropbox in the Project Configuration page.

### **Script Sources**

The Script Sources page can be reached by clicking on the Execution option in the nav-bar, then clicking on the Script Sources option in the drop-down menu. In order to import scripts from GitHub, a GitHub source can be setup in the Script Sources page.

The parameters for setting up an script source is as follows:

**GIT Repo:** Name of the github repo.

**GIT Branch:** Name of repo branch.

**GIT Token:**Authenticated token with clone permissions.

Once the repo has been setup, the repo can be imported by selecting from the Script Source dropbox in the Execution Scripts popup.

# 

### **Executing a Project**

Once the project has been setup, the project can be run from the **Execution Details** page, which can be reached by clicking on the Execution Details option in the Execution dropbox.

If the project is scheduled to run daily, the clock icon will be displayed next to the Run Project icon. For running a project On-Demand, the user must click the Run Project Icon (3 stripes with small inverted triangle).

**Note:**More information on GE parameters for each step can be found in GDM by clicking on the **?**icon located at the right top corner of the respective page.

# Data Feed Documentation

All core data sources are documented for developer and end-user reference. The docs are generated using the Python package sphinx – please see the sphinx documentation (http://www.sphinx-doc.org/en/master/) for details. The current version of the compiled documentation can be found on cloudwatch: <https://d25470mxj4yvu0.cloudfront.net>

## **Content Guidelines:**

For each source, there are two sets of docs generated – user-facing, and development.

### User-facing Docs:

The user-facing docs contain all source and table-specific details (storage information, delivery cadence, schema, sample rows, etc.) for any tables to be accessed by data science end-users. Tables documented here include any Event tables transformed in the level 0 processes, IDMap tables, and Segment tables.

In general, the user-facing docs for each source have four sections: Before You Get Started, Tables, Use Cases, and Development (the last of which just links to the associated development doc).

Before You Get Started:

The Before You Get Started section catalogs all of the general, source-level information of note.

* Overview: Link to sharepoint (can use the exact text below with the link replaced)​
* Critical Info: Any information immediately relevant to data scientists who might potentially work with this data source -- data generating/collection processes, gaps in coverage or skews in distribution, notes about format, usage suggestions, etc.

Tables:  
Under the Tables header, each table accessible to end-users gets its own subsection. Individual table subsection headers take the format Table | <type or name> if the table has been altered at all from the raw schema, and Source | <type or name> if it’s unchanged from the raw schema. Each table subsection in turn has 3 content subsections, detailed below.

* Description: Basic description of the table in a sentence or less.​
* Access: Code block detailing how to access this table via the module. This may change in the near future, so please put a placeholder codeblock in the interim. Example (ComScore Census):​
* Schema and Sample Rows: As the name suggests, this houses the table schema and a handful of sample rows (5 usually) as well as a bulleted list of additional table information. This section is collapsable. Please see the template markdown files for examples of how to create collapsable sections and tables.
  + Fields in the additional info list include Storage Type, Partitioning, File Format, File Size, Start Date, and Delivery Cadence. Example (ComScore Census):​
  + The schema should have the following columns: Name, Type, Description, and Join Key (Table), the last of which indicates if the field is a join key, and if so, what key it joins to and what table that key lives in.​

Use Cases:  
The Use Cases section contains links to any past projects utilizing this dataset in an impactful and/or novel way. Very few sources have the Use Case section populated, and it will not likely be a priority for the GDD team.

Development:  
The Development section in the user-facing docs acts as an access point to the development doc detailed below.

### Development Docs:

The Development docs house both table details for any raw tables not available to the user and documentation for the transformations used to generate user facing tables. This is where the content of any level 0 transformations will live.

Source Tables:  
This section is almost identical to the Tables section found in the user-facing docs, the only differences being two additional fields in the information list, Origin and Location, which refer to the origin (spine prod) and destination (core data) s3 locations, and no Access subsection.

Transformations:  
The transformation section, like the Source Tables section, gives each transformation its own subsection. Transformation subsection headers use the convention Transform | <name> (type). Each has two subsections.

* Description: Brief description of the transformation
* Details: Collapsable list with the following additional subsections. Inputs, which lists the documentation links to all tables involved in the transformation, Row Operations, which details any operation done row-wise on the data during the transformation, Column Operations, which details any operation done column-wise on the data during the transformation, Global Execution Link, which has a link to the repository where the transformation sits, and Outputs, which has a link to the documentation of the outputs of the transformation.

## **Configuration Instructions:**

The basic paradigm is as follows – for each source, we have two markdown files that sphinx will compile to generate the two sets of documentation detailed above. All markdown files for all sources (along with the sphinx machinery) live in the pcs-analytics-doc repository.

The markdown files used to generate the docs live in ./sphinx\_doc/source/data/<source>/. Naming conventions for the markdown files --

* user-facing: <source>-<table>-data.md
* dev: <source>-<table>-dev.md

The README.md in the top level of the repository has detailed instructions on how to build the documentation. If new source directories and markdowns are being added, be sure to add them to the data index.rst,./sphinx\_doc/source/data/index.rst

All markdown formatting for tables, headers, menus, dropdowns, etc. can be lifted from other existing markdowns and/or the template markdowns – our suggestion is to simply copy+paste a markdown and replace the content.