Apache Distill Documentation

Release 0.1.3

Michelle Beard <msbeard@apache.dot.org>

Jan 19, 2022

CONTENTS

Apache Distill is part of the SensSoft software stack. It comprises an analytical engine for SensSoft UserALE.js and SensSoft UserALE.PyQt5 to interact with user activity logs and apply basic analtical operations to the data.

Install instructions can be found here.

A contribution guide has been provided here.

CHAPTER

ONE

USER'S GUIDE

1.1 Installation Guide

1.1.1 Installing Apache Distill

The first step is to install Apache Distill. First, checkout the latest version of Apache Distill.

```
$ git clone https://git-wip-us.apache.org/repos/asf/incubator-senssoft-distill.git
```

Apache Distill is a python project, so it can be installed like any other python library. Several operating systems (Mac OS X, Major Versions of Linux/BSD) have Python pre-installed, so you should just have to run

```
$ easy_install distill
```

or

```
$ pip install distill
```

Users are strongly recommended to install Apache Distill in a virtualenv. Instructions to setup an virtual environment will be explained below.

Note: When the package is installed via `easy_install` or `pip` this function will be bound to the `distill` executable in the Python installation's `bin` directory (on Windows - the `Scripts` directory).

1.1.2 Installing Apache Distill in an Virtual Environment

virtualenv is a tool to create isolated Python environments. virtualenv creates a folder which contains all the necessary executables to use the packages that the Apache Distill project would need.

Install virtualenv via pip:

```
$ sudo env/bin/pip install virtualenv
```

Start by changing directory into the root of Apache Distill's project directory, and then use the virtualenv command-line tool to create a new environment:

```
$ mkdir env
$ virtualenv env
```

Activate environment:

```
$ source env/bin/activate
```

Install Apache Distill requirements:

```
$ env/bin/pip install -r requirements.txt
```

To build the source code and run all unit tests.

```
$ env/bin/python setup.py develop test
```

Launch local Apache Distill server, running on localhost:8090:

```
$ env/bin/dev
```

Deactivate environment

```
$ deactivate
```

1.1.3 Running Apache Distill on Docker Compose

From the project directory, start up Apache Distill in the background.

```
$ docker-compose up -d
Starting elastic
Starting logstash
Starting kibana
Starting distill
$ docker-compose ps
Name                Command                State                      Ports
-------------------------------------------------------------------------------------
↪---------
distill    /bin/sh -c python distill/ ...   Up      0.0.0.0:8090->8090/tcp
elastic    elasticsearch                    Up      0.0.0.0:9200->9200/tcp, 0.0.0.0:9300-
↪>9300/tcp
kibana     /tmp/entrypoint.sh               Up      0.0.0.0:5601->5601/tcp
logstash   logstash -f /etc/logstash/ ...   Up
```

To stop services once you've finished with them:

```
$ docker-compose stop
```

1.1.4 Deployment with Nginx and Gunicorn

I will describe a setup with nginx as a web server on Ubuntu. A web server cannot communicate directly with a Flask application such as Apache Distill. Thus gunicorn will be used to act as a medium between the web server and Apache Distill. Gunicorn is like an application web server that will be running behind nginx, and it is WSGI compatible. It can communicate with applications that support WSGI – Flask, Django, etc.

Install requirements.

```
$ sudo apt-get update
$ sudo apt-get install -y python python-pip nginx gunicorn
```

Create a directory to store the project.

```
$ sudo mkdir /home/pubic_html && cd /home/public_html
```

Download the project from the GitHub repository and copy the application to the `/home/public_html` directory.

```
$ git clone https://git-wip-us.apache.org/repos/asf/incubator-senssoft-distill.git /home/
↪public_html
```

Install Apache Distill's requirements either globally or in a virutal environment:

```
$ env/bin/pip install -r requirements.txt
```

Apache Distill has provided an nginx configuration file located in `distill/deploy/nginx.conf`.

Gunicorn will use port 8000 and handle the incoming HTTP requests.

Restart nginx to load the configuration changes.

```
$ sudo /etc/init.d/nginx restart
```

Run gunicorn on port 8000.

```
$ gunicorn --workers 4 --bind unix:distill.sock -m 007 deploy/run_server:app
```

Start a new browser instance and navigate to http://localhost.

1.1.5 Installing Documentation

To save yourself the trouble, all up to date documentation is available at https://draperlaboratory.github.io/distill/.

However, if you want to manually build the documentation, the instructions are below.

First, install the documentation dependencies:

```
$ env/bin/pip install -r doc_requirements.txt
```

To build Apache Distill's documentation, create a directory at the root level of `/distill` called distill-docs.

```
$ mkdir distill-docs & cd distill-docs
```

Execute build command:

```
# Inside top-level docs/ directory.
$ make html
```

This should build the documentation in your shell, and output HTML. At then end, it should say something about documents being ready in `distill-docs/html`. You can now open them in your browser by typing

```
$ open distill-docs/html/index.html
```

1.2 Quickstart Guide

1.2.1 Usage

Using curl:

```
$ curl -XGET 'http://localhost:8090/app/register' -d '{
        "application_name" : "my_app",
        "version" : "0.1",
        "application_description" : "my test app"
}'
```

CHAPTER

TWO

API REFERENCE

This entire section is mainly for Developers of Apache Distill. This section was automatically generated by Sphinx and apidoc.

2.1 API Documentation

2.1.1 Apache Distill HTTP Client

RESTful Endpoints

The flask object implements a WSGI application and acts as the central object. It is passed the name of the module or package of the application. Once it is created it will act as a central registry for the view functions, the URL rules, template configuration and much more.

The name of the package is used to resolve resources from inside the package or the folder the module is contained in depending on if the package parameter resolves to an actual python package (a folder with an *__init__.py* file inside) or a standard module (just a *.py* file).

For more information about resource loading, see `open_resource()`.

Usually you create a `Flask` instance in your main module or in the *__init__.py* file of your package like this:

```
from flask import Flask
app = Flask(__name__)
```

About the First Parameter

The idea of the first parameter is to give Flask an idea what belongs to your application. This name is used to find resources on the file system, can be used by extensions to improve debugging information and a lot more.

So it's important what you provide there. If you are using a single module, *__name__* is always the correct value. If you however are using a package, it's usually recommended to hardcode the name of your package there.

For example if your application is defined in *yourapplication/app.py* you should create it with one of the two versions below:

```
app = Flask('yourapplication')
app = Flask(__name__.split('.')[0])
```

Why is that? The application will work even with *__name__*, thanks to how resources are looked up. However it will make debugging more painful. Certain extensions can make assumptions based on the import name of your application. For example the Flask-SQLAlchemy extension will look for the code in your application that triggered an SQL query in debug mode. If the import name is not properly set up, that debugging information is lost. (For example it would only pick up SQL queries in *yourapplication.app* and not *yourapplication.views.frontend*)

New in version 0.7: The *static_url_path*, *static_folder*, and *template_folder* parameters were added.

New in version 0.8: The *instance_path* and *instance_relative_config* parameters were added.

param import_name the name of the application package

param static_url_path can be used to specify a different path for the static files on the web. Defaults to the name of the *static_folder* folder.

param static_folder the folder with static files that should be served at *static_url_path*. Defaults to the `'static'` folder in the root path of the application.

param template_folder the folder that contains the templates that should be used by the application. Defaults to `'templates'` folder in the root path of the application.

param instance_path An alternative instance path for the application. By default the folder `'instance'` next to the package or module is assumed to be the instance path.

param instance_relative_config if set to *True* relative filenames for loading the config are assumed to be relative to the instance path instead of the application root.

2.1.2 Apache Distill Analytics

Graph Analytics

`class distill.algorithms.graphs.graph.`**`GraphAnalytics`**

Bases: `object`

Distill's graph analytics package. Apply graph algorithms to User Ale log data segmented with Stout.

`static` **`foo`**`()`

Statistics Package

2.1.3 Apache Distill Models

Brew Interface

`class distill.models.brew.`**`Brew`**

Bases: `object`

Distill supports basic CRUD operations and publishes the status of an persistenct database. Eventually it will support ingesting logs sent from an registered application.

`static` **`create`**`(`*app*`)`

Register a new application in Distill

```
{
        "application" : "xdata_v3",
        "health" : "green",
        "num_docs" : 0,
        "status" : "open"
}
```

Parameters `app` – [string] application name (e.g. xdata_v3)

Returns [dict] dictionary of application and its meta information

static delete(*app*)

Technically closes the index so its content is not searchable.

Parameters `app` – [string] application name (e.g. xdata_v3)

Returns [dict] status message of the event

static get_applications()

Fetch all the registered applications in Distill.

Note: Private indexes starting with a period are not included in the result set

Returns [dict] dictionary of all registered applications and meta information

static get_status()

Fetch the status of the underlying database instance.

Returns [bool] if connection to database instance has been established

static read(*app*, *app_type=None*)

Fetch meta data associated with an application

```
Example:
{
        "application" : "xdata_v3",
        "health" : "green",
        "num_docs" : "100",
        "status" : "open"
        "types" : {
                "raw_logs" : {
                        "@timestamp" : "date",
                        "action" : "string",
                        "elementId" : "string"
                },
                "parsed" : {
                        "@timestamp" : "date",
                        "elementId_interval" : "string"
                },
                "graph" : {
                        "uniqueID" : "string",
                        "transition_count" : "long",
                        "p_value" : "float"
                }
        }
}
```

Parameters `app` – [string] application name (e.g. xdata_v3)

Returns [dict] dictionary of application and its meta information `static update(app)`

Todo: Currently not implemented

Stout Interface

`class distill.models.stout.Stout`

Bases: `object`

Main Stout class to support ingest and search operations.

`static ingest()`

Ingest data coming from Stout to Distill

`class distill.models.stout.StoutDoc(meta=None, **kwargs)`

Bases: `elasticsearch_dsl.document.DocType`

Representation of a Stout documentat.

`save(*args, **kwargs)`

Save data from parsing as a Stout document in Distill

UserAle Interface

`class distill.models.userale.UserAle`

Bases: `object`

Main method of entry to perform segmentation and integration of STOUT's master answer table (if STOUT is enabled). Advanced and basic analytics is performed in the distill.algorithms.stats and distill.algorithms.graphs module.

`static denoise(app, app_type='parsed', save=False)`

`static search(app, app_type=None, filters=[], size=100, include='*', scroll=None, sort_field=None)`

Perform a search query.

Parameters

- **app** – [string] application id (e.g. "xdata_v3")
- **app_type** – [string] name of the application type. If None all application types are searched.
- **filters** – [list of strings] list of filters for a query.
- **size** – [int] maximum number of hits that should be returned
- **sort_field** – [string] sorting field. Currently supported fields: "timestamp", "date"

Returns [dict] dictionary with processed results. If STOUT is enabled, STOUT data will be merged with final result.

`static segment(app, app_type=None, params='')`

Just support match all for now.

`distill.models.userale.merge_dicts(lst)`

`distill.models.userale.parse_query_parameters(indx, app_type=None, request_args={})`

2.1.4 Apache Distill Utilities

Query Builder

class `distill.utils.query_builder.`**`QueryBuilder`**(*query=None*)

Bases: `object`

`add_filters`(*filters*)

`add_sorting`(*sort_field=''*, *sort_order=''*)

Exception Handling

exception `distill.utils.exceptions.`**`Error`**

Bases: `Exception`

Base class for exceptions.

exception `distill.utils.exceptions.`**`ValidationError`**(*url*, *msg*)

Bases: *`distill.utils.exceptions.Error`*

Exceptions raised for errors in validated a url.

Validation Library

`distill.utils.validation.`**`str2bool`**(*v*)

Convert string expression to boolean

Parameters **v** – Input value

Returns Converted message as boolean type

Return type bool

`distill.utils.validation.`**`validate_request`**(*q*)

Parse out request message and validate inputs

Parameters **q** – Url query string

Raises *`ValidationError`* – if the query is missing required parameters

CHAPTER

THREE

ADDITIONAL NOTES

Design notes, legal information and changelog are here.

3.1 Authors

Apache Distill is written and maintained by Draper and various contributors:

3.1.1 Development Lead

- Michelle Beard <msbeard@apache.org>

3.1.2 Additional Staff

- Laura Mariano <lmariano@apache.org>
- Dr. Joshua Poore <jpoore@apache.org>
- Clay Gimenez <cgimenez@dapache.org>
- Steven York <syork@draper.com>

3.2 Contributing to Apache Distill

Thank you for contributing to the Apache Distill project!

There are certain procedures that must be followed for all contributions. These procedures are necessary to allow us to allocate resources for reviewing and testing your contribution, as well as to communicate effectively with you during the review process.

1) Create an issue in JIRA

   All changes to Apache Distill must have a corresponding issue in JIRA so the change can be properly tracked:

   https://issues.apache.org/jira/browse/senssoft

   If you do not already have an account on JIRA, you will need to create before creating your new issue.

2) Make and test your changes locally

   The Apache Distill source is maintained in a git repository hosted on Apache:

https://git-wip-us.apache.org/repos/asf/incubator-senssoft-distill.git

To make your changes, fork the repository and make commits to a topic branch in your fork. Commits should be made in logical units and must reference the JIRA issue number:

```
$ git commit -m "#SENSSOFT-123: #High-level message describing the changes."
```

Avoid commits which cover multiple, distinct goals that could (and should) be handled separately.

If you do not already have an account on JIRA, you will need to create one before making your changes.

3) Submit your changes via a pull request on Git

   Once your changes are ready, submit them by creating a pull request for the corresponding topic branch you created when you began working on your changes.

   The Apache Distill team will then review your changes and, if they pass review, your changes will be merged.

3.3 Changelog

3.3.1 0.1.3 (2016-09-19)

- Moved to Apache.
- Updated all documentation.
- Added License headers
- Docker compose file added to assist deployment of ELK stack with Distill

3.3.2 0.1.2 (2016-07-22)

- Moved CRUD operations from UserAle model to Brew model.
- Added API specs to segment UserAle data from Elasticsearch
- Added deployment instructions

3.3.3 0.1.1 (2016-06-14)

- Completed index route for status endpoint which lists all applications registered and their document count segmented by type.
- Updated setup.py to reference deploy scripts
- Example configuration to deploy Distll with Gunicorn and Nginx for Linux/Mac users
- Added UserAle and Stout classes.
- Updated requirements.txt for deployment.

3.3.4 0.1.0 (2016-04-01)

Initial release.

3.4 License

CHAPTER FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX