Testing the Tableau connector for Elasticsearch

This folder contains the files needed to run Tableau's TDVT suite, to validate the connector's applicability.

Testing requirements

Running the TDVT tests require a working instance of Elasticsearch (local or remote) and a Windows desktop with: * a Python 3 installation; * Tableau Desktop, licensed (the trial mode won't suffice for a JDBC connector), past or at 2019.4 release. * the Elastic JDBC driver available.

All these should be at their latest released version.

Note: If running Elasticsearch remotely, both machines must be (1) time-synchronized and (2) have the same locale settings (i.e.: the calendars on both must start on the same day of week, same date format etc.).

Elasticsearch setup

The Elasticsearch server should be on the latest release, have the SQL plugin installed (i.e. the "default" distribution) and a valid license for it (can be a temporary trial license).

No other settings need to be changed from default, with one exception: the inline script compilation rate needs to be elevated from the default (current: 75/5m). This dynamic cluster setting is called `script.context.aggs.max_compilations_rate` starting with 7.9 releases (for aggregations) and `script.max_compilations_rate` up to 7.8. The minimum high-enough value might be hard to get right, but a fast run of the TDVT tests yield a ~7 tests/sec, so with a generous margin, just for testing, one could set a value of `1000/1m`.

The data for running the TDVT suite needs to be loaded into Elasticsearch prior to running the tests. This can be achieved in a few ways:

1. Import using Elastic's ODBC driver testing suite

ODBC's driver test suite is a Python application that, among other things, will load the testing data into a running Elasticsearch instance. It requires a Python 3 installation with requests and psutils PIP modules installed.

Clone it from its GitHub repo and run the following from within the top repo directory:

```
python3 ./test/integration/ites.py -p http://user:password@host:port -tx
```

where the `user`, `password`, `host` and `port` are the credentials and URL to access the Elasticsearch instance.

This will download the TDVT CSV data sources from their connector-plugin-sdk repository and upload them to the destination test Elasticsearch instance. (It will also install the needed pipelines, templates/mappings to accomodate the source data, along with other test data irrelevant for TDVT testing – just ignore or delete the other created indices).

2. Reindex from remote

Note: this requires some familiarity with Elasticsearch.

If there is an available other Elasticsearch instance running, that has the test data loaded already, this data can be imported through the `_reindex` API. Follow the two steps detailed in Reindex from a remote cluster.

<TODO: detailed sequence of steps>

3. Import using Elastic's Logstash application

Logstash is a Java application that can ingest data from various sources and in various formats and upload them into an Elasticsearch instance.

Note that Logstash will add a few extra fields per row ("document" in Elasticsearch lingo); these shouldn't interfere with the testing, however.

1. Download TDVT's data files Calcs.csv and Staples_utf8.csv and place them into a directory that will be reachable by Logstash.
2. Create the Elasticsearch indices `calcs` and `staples` and use for them the mappings under these links: calcs and stapes. Create an ingest pipeline called `calcs-pipeline` with the definition here.
3. Adapt the config file under the `logstash` folder, updating the , and tags in it.
4. Relaunch Logstash using the updated config file in previous step and wait until the files have been ingested. `calcs` index will need to have 17 documents and `staples` 54860.

Running TDVT

Automated

1. Place Elasticsearch JDBC driver into Tableau's driver folder under: `C:\Program Files\Tableau\Drivers`.
2. Place the `.taco` file either in Tableau's dedicated connectors directory, `C:\Users\[Windows User]\Documents\My Tableau Repository\Connectors`, or a custom one ("").

3. Use the `tdvt_run.py` application, that clone the TDVT SDK repo, setup config files and launch the TDVT run: `python3 ./tdvt_run.py -u "http://user:pass@elastic-host:9200" -t <taco dir path>`

Manually

Setting up the TDVT testing involves following the steps detailed in the official documentation. The "fragment" in parantheses reference the respective chapters in the documentation. It is recommended to execute each test run starting afresh.

1. Same as in the automated testing.
2. Create new Tableau data sources for the `calcs` and `Staple` tables (#`Test a new data source`), or, alternatively, use those available already in this repo. To set up new sources, launch Tableau from command line with the following parameters (PowerShell example): `.\tableau.exe -DConnectPluginsPath=<path> -DDisableVerifyConnectorPluginSignature=true` where `<path>` is either the path to the directory containing the `.taco` connector *or* the path to the directory containing the connector directory, if this isn't yet packaged.

   Note: When connecting, make sure you pass the `timezone=Z` parameter into the `Additional settings` field of the connection dialog. This sets the timezone for the time data to UTC; if this isn't set, the JDBC driver will use JVM's/system's time zone, which will then result in some failed tests if the machine's not set to the UTC timezone.

   Save the TDS files as `cast_calcs.elastic.tds` and `Staples.elastic.tds`
3. Setup a TDVT "workspace" (#`Set up`), i.e. a directory containing the test files. Either package TDVT and install it as a Python PIP module (recommended, if working), or simply copy the `tdvt` directory of the repo into the "workspace" directory. Invoking the TDVT will then be done as `py -3 -m tdvt.tdvt <params>`, or `py -3 .\tdvt\tdvt_launcher.py <params>`, respectively. In the steps below the invokation will be indicated by the `$TDVT` call. `$TDVT action --setup`

   Copy/move the above saved `*.tds` files into the just created `tds` directory in the workspace. Edit `config/tdvt/tdvt_override.ini` to update `TAB_CLI_EXE_X64` definition.
4. Generate the tests by invoking TDVT as follows:

   `$TDVT action --add_ds elastic`

   When asked for a password, use the same value as in step "2." above when connecting to Elasticsearch. For the "logical query config" use `simple_lower`.

Edit the `elastic.ini` file in the `config` directory in the workspace and add the following line under the `[Datasource]` section: `CommandLineOverride = -DConnectPluginsPath=<path> -DDisableVerifyConnectorPluginSignature=true`, where `<path>` has the same value as in step "2.".

5. Run the tests:

   ```
   $TDVT run elastic
   ```

   Note: If running on a busy machine, TDVT's thread allocation can be throttled with the `-t <threads>` argument, where should take the value of available CPU execution units, or even `1` if running on a slower VM.