**How to Deploy JanusGraph data Loader in any Cluster and start loading data?**

**Limitations:** Currently, the Schema to load into Graph is generated manually (Can be updated at later stage to generate dynamically based on schema files, but isn’t priority for now). So, the data loading in any cluster needs to adhere to that Schema, and a new Source addition would mean updating the Gremlin Queries which generate those Schema manually. This drawback will be taken care of when Production ready code is deployed (Scala code)

**Pre-Requisites**:

1. Anaconda is Installed on system, and a Conda environment with Python >= 3.5 is configured in that environment. For example, we use env named “stormgraph” and create it as follows:
   1. conda create -n stormgraph python=3.5
2. Once anaconda environment is created, we will need to install all dependencies needed to load the data. To install dependencies, a “requirements.txt” file is provided with Python library/module and can be used to port across the dependencies as follows:
   1. Activate the Conda env: “conda activate stormgraph”
   2. Goto directory: “cd sia/stormgraph/src/main/python”
   3. Install dependencies: “pip install -r requirements.txt”
3. Once the dependencies are installed, please be noted that it still doesn’t install “JanusGraph-Python” library. That is my own developed custom library for Open Community which is getting reviewd and hence not released. The same is provided as part of the repository, and can be found under “resources” with name “janusgraph\_python-0.3.0.1.tar.gz”
4. To install JanusGraph-Python library, simply goto “resources” folder and do **“pip install janusgraph\_python-0.3.0.1.tar.gz”** or **“pip install janusgraph\_python-0.3.0.1-py3-none-any.whl”** and it installed JanusGraph Python into “stormgraph” env.
5. Now that we have all dependencies installed, all the pre-requisite for the code to work is done, but that is not all.

**Running Data Loader:**

1. Now that we have understood about configurations needed from Application side to connect to JanusGraph and Load data in it, but before we do that, we will need to configure JanusGraph first. (Note this if first step to loading data, and without this data loading will fail)
   1. Install JanusGraph on cluster, Unzip it, and create a new configuration with changes as described bellow. For example we create the updated file with name “test.properties” under “conf” directory.
   2. Create sample properties file referencing new Graph:

|  |
| --- |
| gremlin.graph=org.janusgraph.core.JanusGraphFactory  storage.backend=hbase  storage.hostname=10.10.110.124,10.10.110.125,10.10.110.127,10.10.110.126  storage.hbase.table=janusgraph\_demo3  storage.hbase.ext.zookeeper.znode.parent=/hbase-unsecure  cache.db-cache = true  cache.db-cache-clean-wait = 20  cache.db-cache-time = 180000  cache.db-cache-size = 0.5  index.search.backend=solr  index.search.solr.http-urls=http://10.10.110.125:8983/solr  index.search.solr.mode=cloud  index.search.solr.zookeeperurl=  10.10.110.124:2181/solr,10.10.110.125:2181/solr,10.10.110.127:2181/solr,10.10.110.126:2181/solr  index.search.solr.configset=janusgraph |

* 1. The following needs to be changed when changing Cluster:
     1. **storage.hostname**: This changes to comma separated hostnames where HBase Cluster is running
     2. **index.search.solr.http-urls:** This changes to URL of Solr Master node. It is of format http://<SOLR\_MASTER\_URL>:8983/solr
     3. **index.search.solr.zookeeperurl:** This needs to change into comma separated all Solr Nodes in Cluster
     4. **storage.hbase.table:** This changes to Graph/Table equivalent where we want to Load data. It can either be New Graph or an existing Graph
     5. **Once the above configurations are changed according to Cluster info, we will first have to Load the config in JanusGraph to make sure that the Graph is available for Gremlin Server (Which will be used by Python application to Load data)**
     6. **Open Gremlin Console and Load Graph:**
        1. Open console: **bin/gremlin.sh**
        2. Load Graph: **graph = JanusGraphFactory.open(“conf/test.properties”)**
  2. **Setup Gremlin Server:**
     1. Once the Graph table is setup using Console in above mentioned properties we can configure Gremlin-Server as follows:
     2. **Sample Configuration:**

|  |
| --- |
| host: bddlltesm01e.test.sstech.us  port: 8182  scriptEvaluationTimeout: 3000000  channelizer: org.apache.tinkerpop.gremlin.server.channel.WsAndHttpChannelizer  graphs: {  graph: conf/janusgraph-py1-hbase-solr.properties  }  scriptEngines: {  gremlin-groovy: {  plugins: { org.janusgraph.graphdb.tinkerpop.plugin.JanusGraphGremlinPlugin: {},  org.apache.tinkerpop.gremlin.server.jsr223.GremlinServerGremlinPlugin: {},  org.apache.tinkerpop.gremlin.tinkergraph.jsr223.TinkerGraphGremlinPlugin: {},  org.apache.tinkerpop.gremlin.jsr223.ImportGremlinPlugin: {classImports: [java.lang.Math], methodImports: [java.lang.Math#\*]},  org.apache.tinkerpop.gremlin.jsr223.ScriptFileGremlinPlugin: {files: [scripts/empty-sample.groovy]}}}}  .  .  . |

* + 1. While we aren’t concerned about unmentioned properties, we are only concerned about **graphs** property.
    2. We update it to following to add our new graph specified:
       1. graph1: conf/test.properties
       2. The above adds and exposes Graph1 to Gremlin server and can be accessed accordingly. This becomes the value of property in **JG\_GRAPH** in **config/python\_janusgraph\_config.json**
       3. Once the graph is added, we will need to add its Traversal component as well so that Graph can be accessed. To do that we will need to edit **scripts/empty-sample.groovy** as follows:
       4. We change **globals << [g : graph.traversal()]**  in last line to following:
          1. **globals << [g : graph.traversal(), g1: graph1.traversal()]**
       5. Once done, we can then start **Gremlin-Server using**:
          1. **nohup bin/gremlin-server.sh conf/gremlin-server/gremlin-server.yaml > server.log &**

**Loading Data**:

Now that the JanusGraph is configured for new Graph into which we want to load data, the following configuration needs to be done before Data Loading starts:

1. A few configuration needs to be done before data loading can be done. Let us explain what each of those files signify:
   1. **Schema files**: These define the Schema for each source to be inserted into Graph. Since mentioned under **Limitations,** that schema is hardcoded currently, so changing this Files doesn’t make sense for our deployment scenario. These files are as follows. (Please note the name of files. These will be updated based on how data is in QA1):
      1. resources/schema/schema\_sample.json
      2. resources/schema/schema\_msexchange.json
   2. **Data Mapper files**: These define the mappings between the fields in records, and Graph properties. These also define the Vertices & Edges in data, and which fields become part of which Vertices and Relationships. These needs to be modified if the fields/data structure in QA1 is differently structured. If a new source gets added also, and adheres strictly to the hardcoded schema we defied then we just need to create a new Data Mapper file for that source, and we can start loading that data into Graph. These files are as follows:
      1. resources/python-schema/datamapper\_sample.json
      2. resources/python-schema/datamapper\_msexchange\_sample.json
   3. **Config File:** This config files defines the connection to JanusGraph and can be configured as follows:
      1. **Sample file:** config/python\_janusgraph\_config.json
      2. **Properties:**
         1. JG\_URL: The URL where JanusGraph server is running
         2. JG\_PORT: The Port where JanusGraph is listening to i.e. default to “8182”
         3. JG\_GRAPH: The configured graph name in Gremlin-Server where data needs to be loaded.
         4. JG\_TRAVERSAL: The Configured traversal name which the Python application will use to load data into Graph. This is dependent on **JG\_GRAPH** param, and will be explained how to configure in next section.

Now that the schema files, datamapper files, config files are updated, please run the following command to start Data Loading:

1. **Loading Data: (Data in my env is under directory /root/data/regenerate. You will need to change reference for all those files based on path in Cluster you are loading data.**

bash **data\_loader\_poc.sh** -m **../resources/schema/datamapper\_sample.json**" -s “**../resources/schema/schema\_sample.js**

**on**" -c “**../resources/config/python\_janusgraph\_config.json**" -l “**../resources/data/sample.records**"

1. **The essential structure of shell script is:** bash data\_loader\_poc.sh -d “<**DATA\_SOURCE\_NAME>**” -m “**<PATH\_DATA\_MAPPER\_FILE>” -**s “**<PATH\_TO\_SCHEMA\_FILE>”** -c **“<PATH\_TO\_JANUSGRAPH\_CONFIG\_FILE>**” -l “**<PATH\_TO\_DATA\_FILE>”**

**WHAT IS SCHEMA FILE?:**

**Defines the nodes and edge types/labels to be added to graph, property of each element and their data types and what to and what not to index.**

**WHAT IS DATAMAPPER FILE?**

**DEFINES HOW THE MAPPING IS BETWEEN SCHEMA/GRAPH ELEMENTS and DATA JSON FILE**