Reading from Janusgraph DB

Services, resources, tosca type definitions, users etc are all stored by SDC in a janusgraph DB in the sdctitan table in the cassandra DB. The following details how the information stored in the janusgraph can be viewed

1. Run janusgraph server
    a. This **docker-compose file** can be used to start up a janusgraph server can then be used to read from the DB
        - set your ip address in the docker-compose file
            - If you are using a local DB use the following command to get your local ip
                - ip route get 8.8.8.8 | awk '/src/{ print $7 }'
        - docker-compose -f docker-compose-cql-es.yml up
2. Start gemlin
    a. Login to the janusgraph server
        - docker exec -it jce-janusgraph bash
    b. Start gremlin
        - bin/gremlin.sh
3. Read
    a. Connect to the graph
        - :remote connect tinkerpop.server conf/remote.yaml
    b. Use gremlin query commands to execute queries to the DB, for example (note the ":>" is part of the command):
        - :> g.V() - list all vertices
        - :> g.V().has('name', 'myService') - list vertices with name 'myService'
        - :> g.V().has('name', 'myService').valueMap() - show details of vertices with name 'myService'
        - :> g.V().has('name', 'myService').properties() - properties of vertices with name 'myService'
        - :> g.V().has('name', 'myService').properties('metadata').value() - value of the property "metadata" of vertices with name 'myService'
        - :> g.V().has('nodeLabel', 'dataType') - data type vertices
            - :> g.V().has('nodeLabel', 'dataType').count() - total of data type vertices
            - :> g.V().has('name', 'ericsson.datatypes.nfv.InstantiateVnfOperationAdditionalParameters').properties()
        - node types (VFCs)
            - All node types
                - :> g.V().has('nodeLabel', 'node_type')
            - All nodes that derives from **tosca.nodes.Root**
                - :> g.V().has('nodeLabel', 'node_type').has('toscaResourceName', 'tosca.nodes.Root').in('DERIVED_FROM')
        - model
            - Model with given name
                - :> g.V().has('name', 'ETSI SOL001 v2.5.1')
            - Model with given id
                - :> g.V().has('uid', 'model.ETSI SOL001 v2.5.1')
            - All models
                - :> g.V().has('nodeLabel', 'model')
            - All Service Templates (Service, VF, PNF, etc.) that is associated to the model
                - :> g.V().has('uid', 'model.ETSI SOL001 v2.5.1').out('MODEL_ELEMENT').has('nodeLabel', 'topology_template')
4. Export
    a. The entire graph can be exported for viewing with a compatible tool by using the following gremlin command which exports to a file with the given name
        - :> graph.io(graphml()).writeGraph("export.xml")
5. View
    a. Cytoscape can be used to view the exported graph. It can be downloded from:
        - https://cytoscape.org/
    b. Once running, select the below to import the graph:
        - File -> Import -> Network From File