System Architecture

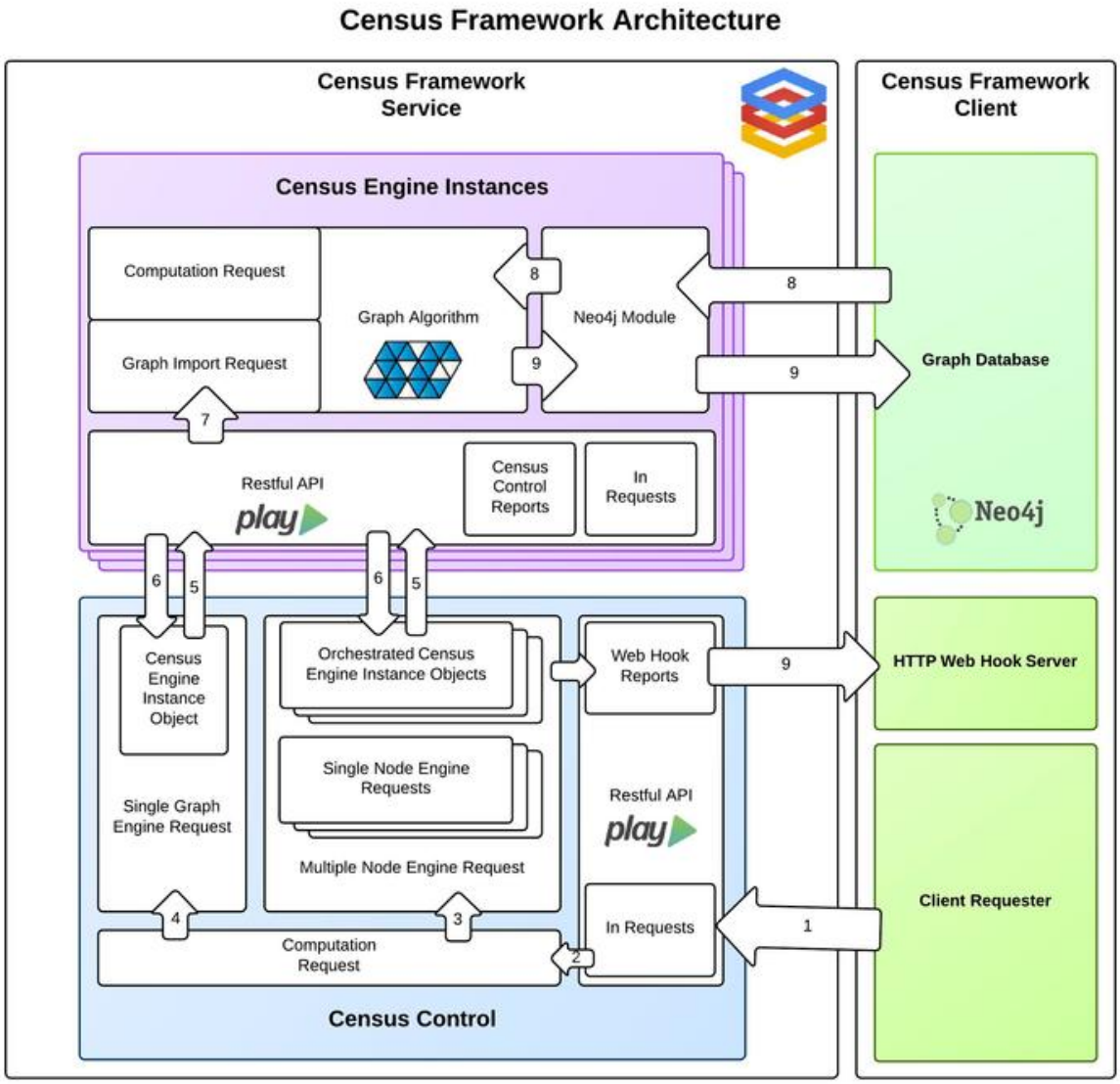
The system is setup on GCE (Google Compute Engine), Census Framework is made of two sub projects, Census Control and Census Engine.

Census Control initializes on a single GCE virtual machine, it is the service which receives computation petitions from the client, then it creates and orchestrates as many instances of Census Engine as needed to compute those petitions.

Census Engine initializes on GCE virtual machines, it is the service which imports the graph (from a Neo4j database) and starts an algorithm computation on it, when done it reinserts the generated data on the same database.

The framework used to do the actual graph computation is Signal Collect v2.0, and the framework used to create the RESTful API is Play Framework v2.1.5.

Note: The Akka versions for Signal Collect and Play Framework can cause dependency conflicts, Play Framework v2.1.5 and Signal Collect v2.0 share a compatible version of Akka.



1) The client sends requests to Census Control through the RESTful API, (including registering a web hook (HTTPHook) to receive reports). The client can receive string tokens back, which are used to reference pending petitions.

2) The Play Framework controller InReqeusts initializes a ComputationRequest object, which saves the request metadata, including the N4j (Neo4j) service instance, and through the Receiver trait, it can initiate an EngineRequest.

3) In case of an all pair algorithm (an algorithm which needs to be computed for every pair of nodes on the graph) or an algorithm which has to be computed individually for every node (like a single source shortest path) a MultiNodeRequest algorithm is created, which creates a SingleNodeRequest for every node on the graph.

Note: Here a query from Census Control to the Neo4j service (this interaction doesn't appear on the architecture diagram) is required to retrieve the nodes id and possible extra data.

The `MultiNodeRequest` also creates an Orchestrator, which creates as many Census Engine Instances as needed to distribute the `SingleNodeRequests` through them.

Note: The `MultiNodeRequest` implements the Receiver trait so that the `ComputationRequest` can initialize it, and each `SingleNodeRequest` implements the Sender trait so that the `Orchestrator's` `Instances` can send the actual requests to the multiple Census Engine services.

4) **Not yet implemented:** In case of a single graph request (an algorithm which can be computed on a single instance, on a single run, like page rank) a SingleGraphRequest is created, which creates a single Instance to start the computation.

Note: The setup of the algorithms on Census Control is in the library package, and the actual implementation of the algorithms on Census Engine is in the library package.

5) Through the `Sender` trait, the `Instances` can send http requests to the Census Engine InRequests module.

6) Through Census Engine's CensusControl module, reports of success or error are sent back to Census Control's InReports module.

7) Before making a graph computation on Census Engine a GraphImportRequest must be done. After that a ComputationRequest can be done for all the requests that uses the same graph for the same algorithm.

8) The N4j module is used to query the client's graph database for the graph importation.

9) When the computation is finished, the generated data is reinserted on the client's Neo4j databse, and Census Control is notified.