Kubernetes Setup

This includes a folder called "kubernetes_deployment" which includes two deployment setups for cassandra and janusgraph. Two things to note on this configuration:

1. This was built to be run on kubernetes running on docker desktop, so docker images are built first and then injected into kubernetes via the docker registry contained in docker desktop (as described here). It is recommended to use a CI/CD pipeline to build the janusgraph image and deploy that into a docker registry in a production environment.
2. This setup uses a configured graph factory, but because no DNS server is available, it uses the ip address of the single cassandra pod. If using a configured graph factory setup as this outlines, it is highly recommended to use a DNS server so that the cassandra hostnames can be used rather than the IP addresses.

Building Janusgraph Docker Image

Inside of `janusgraph-full-0.6.3`, run the following:

```
docker build -t janusgraphbuild .
```

after building, this image will take in the following environmental variables (examine `docker-entrypoint.sh` for more detail):

1. CASSSANDRA_HOSTNAME: the hostname of the cassandra pod (here injected via kubernetes, but can be used as an environmental variable if using an EC2 instance)
2. CASSANDRA_USERNAME: the username of the primary cassandra account
3. CASSANDRA_PASSWORD: the password of the primary cassandra account

Deploying Services and deployments

Inside of the `kubernetes_deployment` folder, run the following:

```
kubectl apply -f cassandra-persistentvolumeclaim.yaml,cassandra-service.yaml,cassandra-deploym
```

This will bring up a single-node cassandra cluster available for testing. These can be modified (and should be modified) to provision a larger cassandra cluster for production. After that runs, execute the following:

```
kubectl apply -f janusgraph-networkpolicy.yaml,janusgraph-service.yaml,janusgraph-deployment.
```

This will deploy janusgraph. Both commands can be run at the same time, but this guarantees that there are little-to-no restarts of either pod. After running the above, verify that the pods are created:

```
$ kubectl get pod
NAME                           READY    STATUS     RESTARTS    AGE
cassandra-f6449f6cd-424p4      1/1      Running    0           81m
janusgraph-5fb95444df-nbdbd    1/1      Running    0           22s
```

NOTE: the pod names you will see will be different than shown here. The services give the more persistent names/ip address with can be seen as:

```
$ kubectl get services
NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)
cassandra     ClusterIP   10.99.81.233   <none>        7000/TCP,7001/TCP,7199/TCP,9042/TCP,916
janusgraph    ClusterIP   10.99.37.23    <none>        8182/TCP
kubernetes    ClusterIP   10.96.0.1      <none>        443/TCP
```

Setup Graph

Inside of the Janusgraph docker build folder, a file called `GraphSetup.groovy` will be created that can be used to setup the graph for the first time. To setup the graph, get the hostname from `kubectl get pod` and execute into it as:

```
 kubectl exec --stdin --tty janusgraph-5fb95444df-nbdbd -- /bin/bash
```

once inside the pod execute the following:

```
janusgraph-5fb95444df-nbdbd:/app# ./bin/gremlin.sh < GraphSetup.groovy
```

then restart the deployment (so that you can access `toygraph_traversal` as a variable in the gremlin console:

```
kubectl rollout restart deployment janusgraph
```

Execute Gremlin Traversal in Groovy Console

Once again, run `kubectl get pod` to get the name of the janugraph pod:

```
$ kubectl get pod
NAME                          READY   STATUS        RESTARTS   AGE
cassandra-f6449f6cd-424p4     1/1     Running       0          3h11m
janusgraph-57464d5b45-wdfzw   1/1     Running       0          16s
```

using that pod name, execute into the pod to get to the gremlin console:

```
kubectl exec --stdin --tty janusgraph-57464d5b45-wdfzw -- /app/bin/gremlin.sh
```

once in gremlin console, execute the following:

```
gremlin> :remote connect tinkerpop.server conf/remote.yaml session
==>Configured localhost/127.0.0.1:8182-[f287aff0-cabb-4e4b-b882-e5ee19899207]
gremlin> :remote console
==>All scripts will now be sent to Gremlin Server - [localhost/127.0.0.1:8182]-[f287aff0-cabb
```

Now you can play with the toygraph as follows:

```
gremlin> toygraph_traversal
==>graphtraversalsource[standardjanusgraph[cql:[10.109.91.42]], standard]
```

```
gremlin> toygraph_traversal.V().count().next()
==>12
```

A full example with gremlin queries:

```
gremlin> toygraph_traversal.V().has('name', 'saturn').valueMap()
==>{name=[saturn], age=[10000]}
gremlin> toygraph_traversal.V().bothE().limit(10).valueMap()
==>{}
==>{}
==>{place=POINT (23.9 37.700001), time=2}
==>{time=1, place=POINT (23.700001 38.099998)}
==>{place=POINT (22 39), time=12}
==>{reason=no fear of death}
==>{}
==>{reason=loves waves}
==>{}
==>{}
gremlin> neptune_id = toygraph_traversal.V().has('name', 'neptune').id().next()
==>16552
gremlin> toygraph_traversal.V(neptune_id).outE('brother').inV().inE('father').outV().out().ha
==>path[v[16552], e[c9h-crs-b2t-9m0][16552-brother->12456], v[12456], e[d1x-fxk-6c5-9m0][2064
gremlin>
```

Congrats, you are now running a basic installation of janusgraph on kubernetes