Assignment 10: Docker Container version [beta]

[The Docker version of this assignment is new: please be on the lookout for problems with the description below.]

You can work on this assignment in small groups, but write up your submissions individually. Alternate versions: with the Raspberry Pi cluster; with virtual machines.

The actual Cluster that we have for this course is nice, but it has one notable limitation: it actually works.

Part of the selling point of the Hadoop technologies is that they are fault tolerant. It's one thing to read that in the docs, but another to see it actually play out. Ryan will be quite cross if we go into the server room to start unplugging nodes to see what happens.

But we can use Docker containers to create our own mini-cluster…

Container Cluster

Requirements for the computer where you do this: a 4-core processor, 8GB memory, 20GB free disk space. The desktops in the computer lab meet these specs, so you can certainly do this assignment there.

You will need Docker *(https://docs.docker.com/get-started/)* (Ubuntu installation instructions *(https://docs.docker.com/install/linux/docker-ce/ubuntu/)* and Docker Compose *(https://docs.docker.com/compose/)* (the `docker-compose` package in Ubuntu).

Start by cloning the Docker cluster project *(https://csil-git1.cs.surrey.sfu.ca/ggbaker/732-docker-cluster)* which contains configuration recipes for the cluster.

```
git clone https://csil-git1.cs.surrey.sfu.ca/ggbaker/732-docker-cluster.git
```

The deployment depends on the Hadoop and Spark distributions, and has a script to retrieve them (assuming a Linux-like system):

```
./get-sw.sh
```

You can build and start the containers like this:

```
docker-compose build
docker-compose up
```

This will start a cluster with three HDFS DataNodes, and three YARN NodeManagers (workers). The docker-compose configuration also includes a fourth of each, but they don't start by default.

Working with the cluster

There is a little more setup to do for this assignment. Copy a file into the newly-created HDFS like this:

```
docker-compose run gateway hdfs dfs -copyFromLocal bigfile /bigfile1
```

The container `gateway` is not really part of the cluster, but can be used to work with it (like the gateway on our actual cluster). For example, you can start a pyspark shell on the gateway (using the "cluster") with this command:

```
docker-compose run gateway pyspark
```

Web Frontends

- HDFS NameNode: http://localhost:9870/ *(http://localhost:9870/)*

- YARN ResourceManager: http://localhost:8088/ *(http://localhost:8088/)*

Notes on the Simulation

Rather than actually destroying disks and nodes, the instructions below do a couple of things to simulate failure. These are:

- Stopping a container (`docker-compose stop`) stops the Hadoop process(es) and effectively makes the node disappear from the cluster. It is as if the node failed completely (until we restart it). Stopping a container keeps the disks intact.
- Removing a container (`docker-compose rm`) destroys the container **and its storage** so we can simulate drive failure (or completely replacing a node).
- Every time you start a container, it will take up a terminal window (or tab). As you stop/start containers, you're going to need a lot of terminals sitting holding their processes.

Expanding the cluster

Have a look at the YARN and HDFS web frontends, and confirm that there are running with three nodes up.

Let's add one more node to both the YARN and HDFS clusters. You will probably have to run each of these in a separate terminal window (since the processes run in the foreground when started like this):

```
docker-compose run yarn-nodemanager-4 /opt/hadoop/bin/yarn nodemanager
docker-compose run hdfs-datanode-4 /opt/hadoop/bin/hdfs datanode
```

Effectively, these prepresent new nodes in YARN and HDFS with newly-formatted hard drives.

Check in the web frontends that it has appeared as a DataNodes and TaskTrackers. The overall capacity (storage, memory, cores) should have increased appropriately as well.

That's how easy expanding a Hadoop cluster is.

Drive Failure

HDFS stores redundant copies. There are two reasons: (1) so the content is available on multiple nodes as input to jobs, and (2) to provide protection against disk failure. In this scenario, we aren't really getting (1) since the HDFS and YARN processes are running separately, but we get (2) with two copies of each block.

So, what actually happens when a disk fails?

Find a file in the HDFS filesystem: go to the web frontend (http://localhost:9870/ *(http://localhost:9870/)*) → Utilities → Browse the filesystem. When you click a filename, a popup window gives you some metadata, including the list of nodes where that file is stored.

Pick one of the containers where that file is stored. We're going to simulate failure of that disk. For the commands below, I'm going to assume it's `hdfs-datanode-3` that we're failing.

Stop the sacrificial node: this will stop the HDFS processes and probably would have been necessary to replace the drive anyway:

```
docker-compose stop hdfs-datanode-3
```

Have a look at the list of Datanodes in the web frontend. (Keep reloading the page to get updated statuses.) What happens?

After HDFS gives up on the “failed” node, have a look at the number of “Under-Replicated Blocks” on the NameNode front page. What happens to the file you were watching? Was the replication maintained? [?]

Note: the dead-node timeouts have been made very short on the VM cluster. They are typically much longer so something like a network switch reboot doesn't declare everything behind it dead.

Replacement Drive

Now we will destroy and recreate the container, effectively wiping the HDFS data from the disk, as we would see on a newly-provisioned drive:

```
docker-compose rm hdfs-datanode-3
docker-compose up hdfs-datanode-3
```

In the web frontend, you should see the node with the "replaced" drive back in the cluster. It should have no blocks being stored, but be ready as part of the cluster

Aside: You might have noticed that the "add a new node" and "replace a drive" procedures are quite similar. To me that makes it feel like what's in a Hadoop cluster is very ephemeral: the cluster consists of whatever resources happen to be there just at the moment. The YARN/HDFS machinery takes care of the rest.

Computation Failure

Now we will simulate failure of a node *during* a job's execution.

Make sure the cluster is up and running. Start a job that will take long enough to give you some time: this can even by the `pyspark` shell:

```
docker-compose run gateway pyspark
```

… with a job like this:

```
sc.range(10000000000, numSlices=100).sum()
```

Start the job, and once the executors are start getting something done, look at the Spark frontend (http://localhost:8088/ *(http://localhost:8088/)*). Select the running application → ApplicationMaster. You will likely have to **edit the URL** of the ApplicationMaster/Spark Frontend: replace `yarn-resourcemanager` with `localhost`.

Find a node that's doing some work (i.e. one listed in the "Executors" tab with tasks running). Stop that node so any work is left unfinished. If node 3 is the one being killed:

```
docker-compose stop yarn-nodemanager-3
```

Keep watching the attempt page in the frontend (reloading as necessary), and the output of your application in the terminal. How does YARN/Spark respond to the node failing? [?]

Other Things To Try

There are many things our little cluster can do (if slowly). These aren't required for the assignment, but may be interesting.

Pick a (small?) file and decrease its replication factor to one:

```
docker-compose run gateway hdfs dfs -setrep -w 1 output/part-00000
```

What happens if a node containing some of that file disappears for a few minutes (and then rejoins the cluster)? What if the drive it's on "fails"?

Of course, you're free to experiment further with your containers: you can always stop and destroy them (control-C and `docker-compose rm`), and start over.

Shutting Down

You can stop the containers by pressing control-C (in whatever terminals things are running in) and

When you're done with the VMs (and especially if you're on the lab computers, **please free up the disk space**):

```
docker-compose down
docker system prune -f
```

Questions

In a text file `answers.txt`, answer these questions:

1. What happened to the HDFS file when one of the nodes it was stored on failed?
2. How did YARN/MapReduce/Spark behave when one of the compute nodes disappeared?
3. Were there more things you'd like to try with this cluster, or anything you did try that you think should have been in this assignment?

Submitting

Submit your `answers.txt` to Assignment 10 in CourSys.

Updated Fri Oct. 25 2019, 14:43 by ggbaker.