1 HBase Primer

These class notes continue the Hadoop Primer class notes. So you might want to make sure that Hadoop is setup correctly before proceeding.

Hadoop (and HDFS) are mostly engineered for batch processing. HBase is a layer that sits between the client and HDFS (or any reliable data store) that provides real time random access to 'big data'.

2 Downloading HBase

Grab the latest version from: `http://hadoop.apache.org/`

As of this writing, I found: `hbase-1.4.3-bin.tar.gz`

You can unzip that into your home folder:

```
tar -xzf ~/Downloads/hbase-1.4.3-bin.tar.gz
```

This created `/home/alex/hbase-1.4.3`. Now define a variable `HBASE_HOME` to point to it. At the top of `~/.bashrc`, add:

```
export HBASE_HOME=/home/alex/hbase-1.4.3
export PATH="$HBASE_HOME/bin:$PATH"
```

3 Configuring HBase

We need to tell HBASE where to store the data, and keep logs.

This goes into: `$HBASE_HOME/conf/hbase-site.xml`:

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://localhost/hbase</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/home/alex/hbase-1.4.3/zookeeper</value>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>
</configuration>
```

The `hbase.rootdir` tells hbase where to store files. In this case, we're pointing it to the local HDFS. You don't need to create the `/hbase` directory in HDFS (HBase will do that automatically).

Thje `hbase.zookeeper.property.dataDir` tells hbase where to maintain Zookeeper stuff. We need to create that folder:

```
mkdir -p /home/alex/hbase-1.4.3/zookeeper
```

Thje `hbase.cluster.distributed` tells hbase to operate in distributed mode—without this hbase would operate in a single machine mode (the `the hbase.rootdir` could be pointing to a local directory `file:///something`.

4 Running Stuff

Need to start Hadoop, so: Go to `$HADOOP_HOME/sbin` directory, and run:

```
./start-all.sh
```

This starts HDFS, etc.
Then go to `$HBASE_HOME/bin` directory and run:

```
./start-hbase.sh
```

Congratulations, you should now have HBase running.

5 Using HBase

Enter hbase shell:

```
hbase shell
```

From here, you can run all sorts of commands. For one, you can check the status of HBase:

```
hbase(main):005:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 3.0000 average load
```

Ok, a bit of architecture. HBase is a key-value store. Mostly. It organizes data into tables. The "key" (of key-value) is the row-key that identifies a record within the table.

Tables can have one or more column families. Each column family has multiple columns. The row-key is present in each column family. Each column family is stored on disk separately (meaning reading things within the same column family is very fast).

With that in mind, if we treat HBase as a quick hash-table, we would first need to define the structure (table, and column families), then we can put values into indexes that are composed of table, column family, row-key, and value.

For example, to create a customer table, with two column families, one for customer details, another for customer orders:

```
create 'customer', 'details', 'orders'
```

To see that it's there:

```
> list
TABLE
customer
```

We can now add data to these key-value stores:

```
put 'customer','1','details:fname','John'
put 'customer','1','details:lname','Doe'
put 'customer','1','details:email','john.doe@gmail.com'
```

We can see that it worked:

```
> scan 'customer'
ROW  COLUMN+CELL
 1   column=details:email, timestamp=1523907165824, value=john.doe@gmail.com
 1   column=details:fname, timestamp=1523907165762, value=John
 1   column=details:lname, timestamp=1523907165787, value=Doe
1 row(s) in 0.0130 seconds
```

We can also retrieve values:

```
> get 'customer','1',{COLUMN=>'details:email'}
COLUMN          CELL
 details:email  timestamp=1523907165824, value=john.doe@gmail.com
1 row(s) in 0.0230 seconds
```

If we don't specify `{COLUMN=>'details:email'}` it would pull all key-values for that row-key.

There's also a delete key:

```
> delete 'customer','1','details:email'
0 row(s) in 0.0380 seconds
```

And there's also a `deleteall` to delete all cells for a given row-key.

```
> deleteall 'customer','1'
0 row(s) in 0.0110 seconds
```