Big Data For Engineers – Exercises

Spring 2020 – Week 5 – ETH Zurich

HBase

Overiew of this exercise sheet

This exercise consists of two main parts:

- Hands-on practice with your own HBase cluster running in Azure.
- Theory exercises on the architecture of HBase.

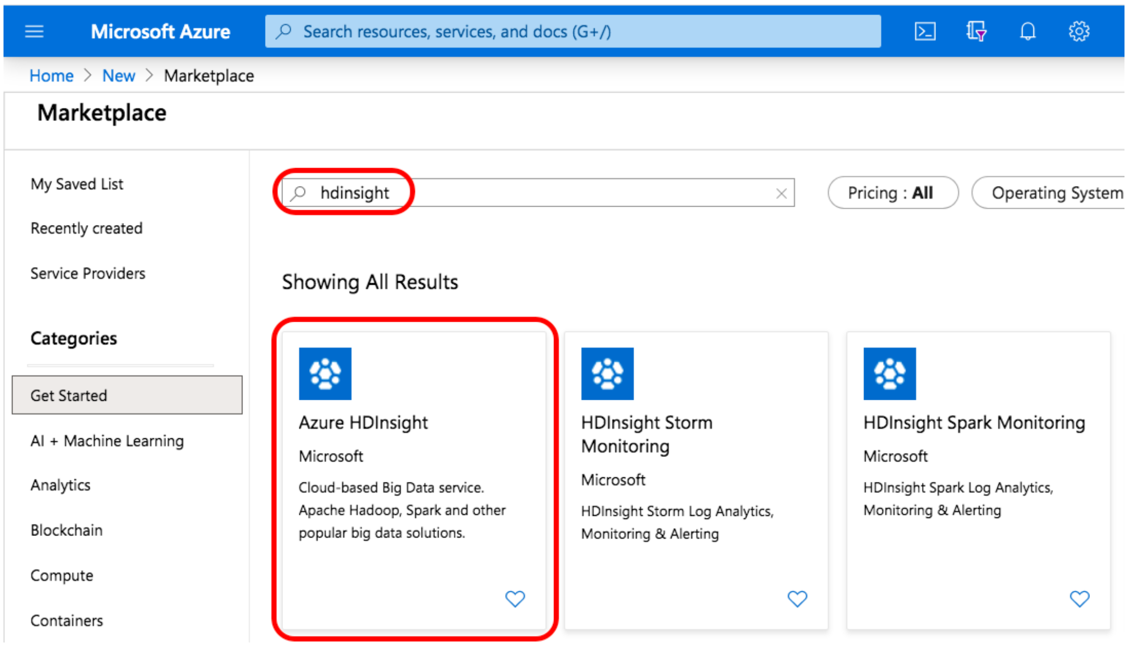
Exercise 1 — Creating and using an HBase cluster

It's time to touch HBase! You will create, fill with data, and query an HBase cluster running on Azure.

Do the following to set up an HBase cluster:

Important: we want you to use a small but real cluster for running HBase rather than a single machine. But, these clusters burn Azure credit very quickly—the cheapest configuration consumes roughly **2 CHF per hour**, which is a lot relative to your overall credit—so it is very important for you to **delete your cluster once you are done.** Luckily, it is possible to keep your data intact when you delete a cluster, and see it again when you recreate it; we will touch upon this in the process. Now, let's start. Those steps are very similar to the HDFS cluster we create on week 3.

1. In Azure portal click the **"Create a resource"** button on the left, type **"hdinsight"** in the search box, and select **"Azure HDInsight"** and click **"Create"**. HDInsight is Microsoft's cloud service which wraps Hadoop, HBase, Spark and other Big Data technologies; read more here.

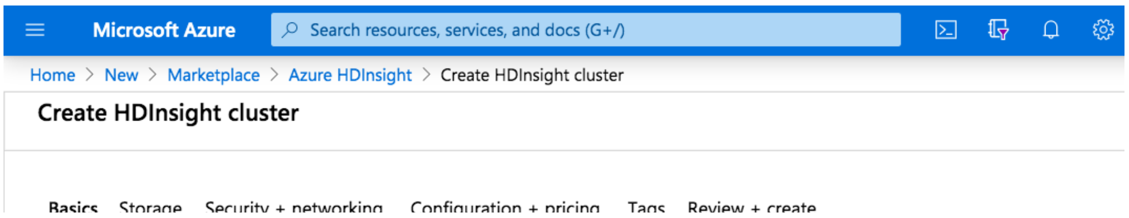


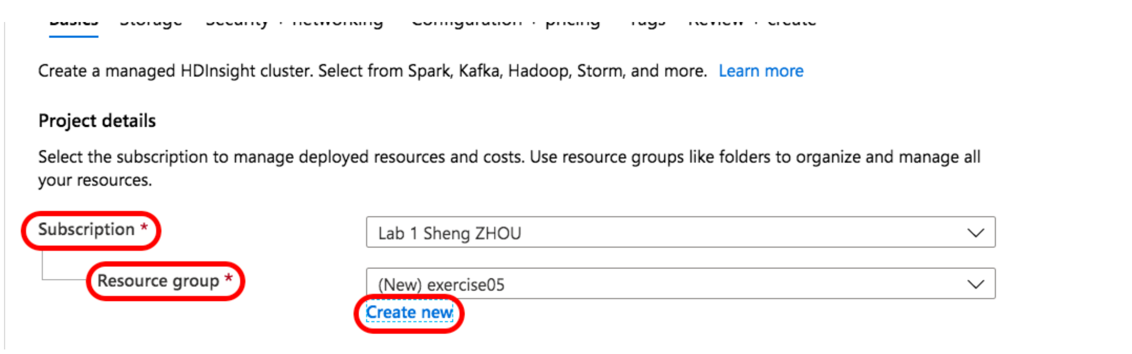
Azure HDInsight
Microsoft
Cloud-based Big Data service. Apache Hadoop, Spark and other popular big data solutions.

HDInsight Storm Monitoring
Microsoft
HDInsight Storm Log Analytics, Monitoring & Alerting

HDInsight Spark Monitoring
Microsoft
HDInsight Spark Log Analytics, Monitoring & Alerting

2. In the **"Basics"** tab, choose a subscription and **create a new resource group** (say "exercise 05").





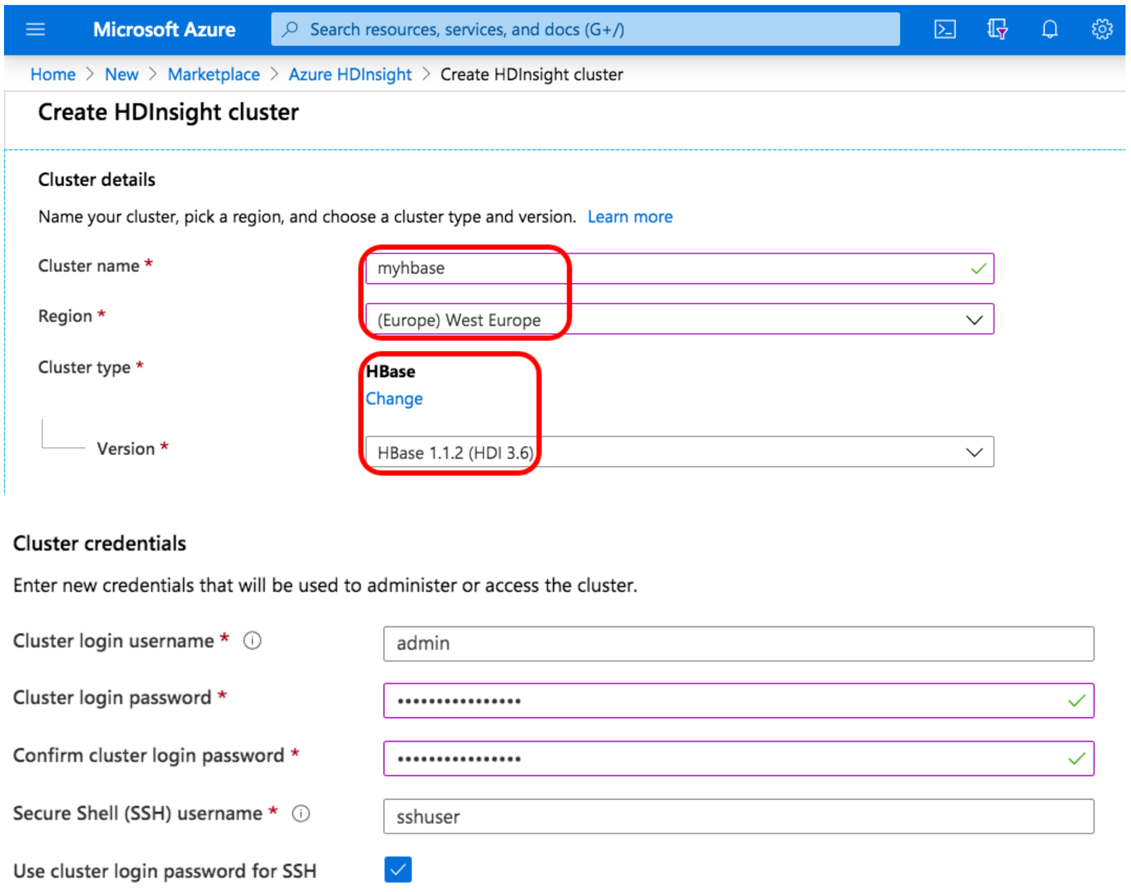
Create a managed HDInsight cluster. Select from Spark, Kafka, Hadoop, Storm, and more. Learn more

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	Lab 1 Sheng ZHOU
Resource group *	(New) exercise05
	Create new

3. Name your HBase cluster and select the region to be **"(Europe) West Europe"**. Choose **"HBase"** for cluster type and use **version 1.1.2**. Then setup the cluster login username and password, as well as the SSH username.



Home > New > Marketplace > Azure HDInsight > Create HDInsight cluster

Create HDInsight cluster

Cluster details

Name your cluster, pick a region, and choose a cluster type and version. Learn more

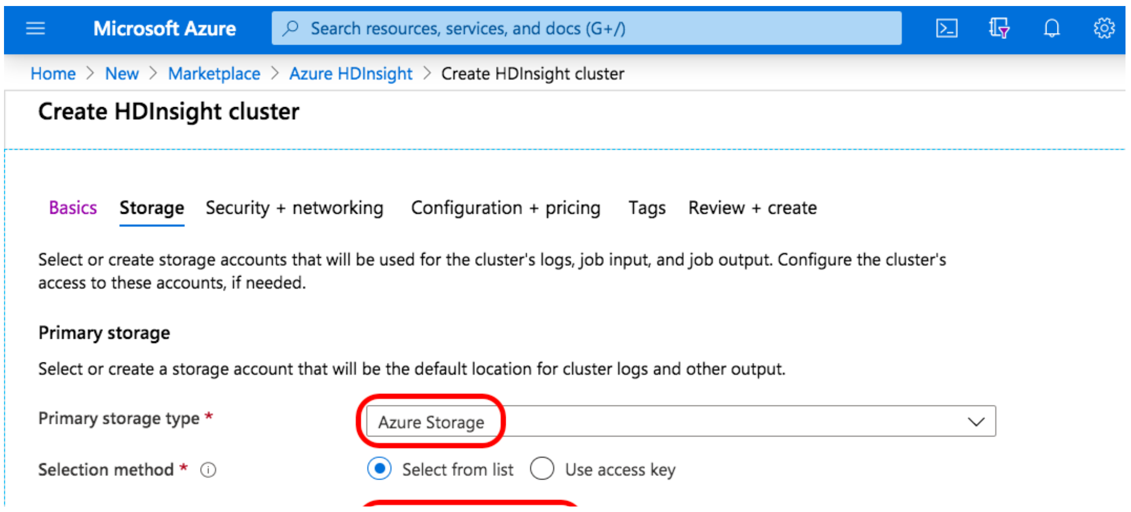
Cluster name *	myhbase
Region *	(Europe) West Europe
Cluster type *	HBase
	Change
Version *	HBase 1.1.2 (HDI 3.6)

Cluster credentials

Enter new credentials that will be used to administer or access the cluster.

Cluster login username *	admin
Cluster login password *	••••••••••••••••
Confirm cluster login password *	••••••••••••••••
Secure Shell (SSH) username *	sshuser
Use cluster login password for SSH	✓

4. Next, we need to configure the **"Storage"** tab. The canonical way would be to use an HDFS cluster as a storage layer for an HBase cluster, but we will be using the Blob service of Windows Azure Storage for this purpose. This has a significant advantage of allowing you to delete your HBase cluster without losing the data: you can recreate the cluster using the same Azure Storage Account and the same container and you will see the same data. This is useful, for example, if you don't have time to finish this exercise in one sitting: you can just delete your cluster, recreate it later, and continue your work. Azure storage is selected by default (see the screenshot).
To setup your HBase cluster for the first time, in **"Primary storage account"** click **"Create new"** and specify a name. Leave everything else as it is and click "Next".



Home > New > Marketplace > Azure HDInsight > Create HDInsight cluster

Create HDInsight cluster

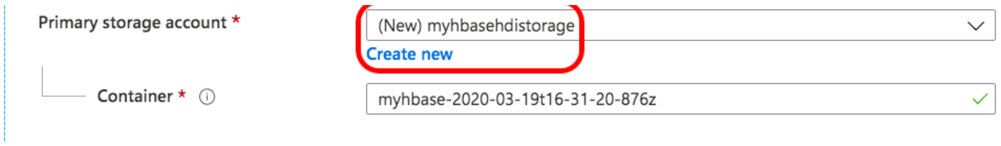
Basics | Storage | Security + networking | Configuration + pricing | Tags | Review + create

Select or create storage accounts that will be used for the cluster's logs, job input, and job output. Configure the cluster's access to these accounts, if needed.

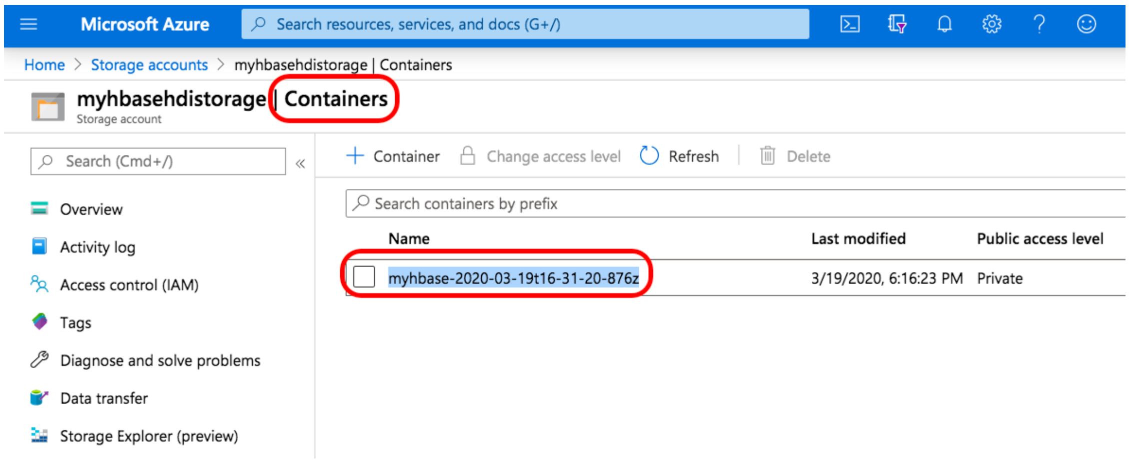
Primary storage

Select or create a storage account that will be the default location for cluster logs and other output.

Primary storage type *	Azure Storage
Selection method *	[✓ Select from list] Use access key

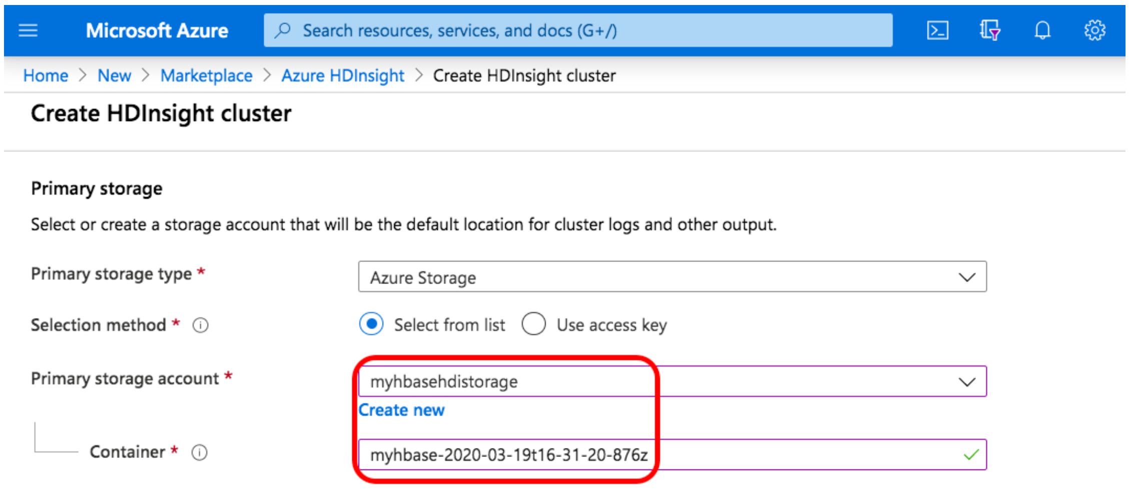

Primary storage account *
(New) myhbasehdistorage
Create new
Container *
myhbase-2020-03-19t16-31-20-876z

Important: if you are recreating your HBase cluster and want to see the existing data, then choose **"Select existing"** and set the container name to the one that you used last time—by default Azure generates a new container name every time you create a cluster, which then points to a different container. The container name can be found in "Storage account - containers".


Microsoft Azure
Search resources, services, and docs (G+/)
Home > Storage accounts > myhbasehdistorage | Containers
Search (Cmd+/)
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Data transfer
Storage Explorer (preview)
+ Container
Change access level
Refresh
Delete
Search containers by prefix

myhbasehdistorage | Containers
Storage account

Name	Last modified	Public access level
myhbase-2020-03-19t16-31-20-876z	3/19/2020, 6:16:23 PM	Private


Microsoft Azure
Search resources, services, and docs (G+/)
Home > New > Marketplace > Azure HDInsight > Create HDInsight cluster
Create new

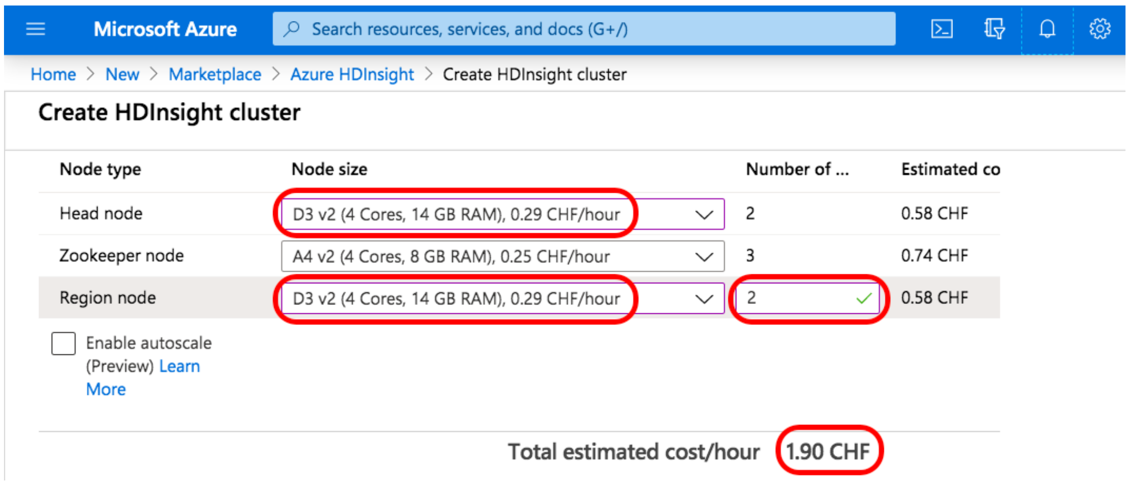
Create HDInsight cluster

Primary storage

Select or create a storage account that will be the default location for cluster logs and other output.

Primary storage type * Azure Storage
Selection method * [✓ Select from list] Use access key
Primary storage account * myhbasehdistorage
Container * myhbase-2020-03-19t16-31-20-876z

5. In the "Security + networking" tab do not choose anything, just click "Next: Configuration + pricing".
6. Now we need to choose the configuration of the nodes in our HBase cluster. It will be enough to have **only 2 RegionServers** (see the screenshot). As for the node size, let us be wise and select the economical option: click on "Region node size" and choose **"D3 V2"**; do the same for the Head nodes; the "Zookeeper" nodes should have **"A4 v2"** selected by default (Zookeeper is a distributed coordination service used by HBase). Click **"Review + create"**.


Microsoft Azure
Search resources, services, and docs (G+/)
Home > New > Marketplace > Azure HDInsight > Create HDInsight cluster

Create HDInsight cluster

Node type	Node size	Number of ...	Estimated co
Head node	D3 v2 (4 Cores, 14 GB RAM), 0.29 CHF/hour	2	0.58 CHF
Zookeeper node	A4 v2 (4 Cores, 8 GB RAM), 0.25 CHF/hour	3	0.74 CHF
Region node	D3 v2 (4 Cores, 14 GB RAM), 0.29 CHF/hour	2	0.58 CHF

Enable autoscale (Preview) Learn More

Total estimated cost/hour 1.90 CHF

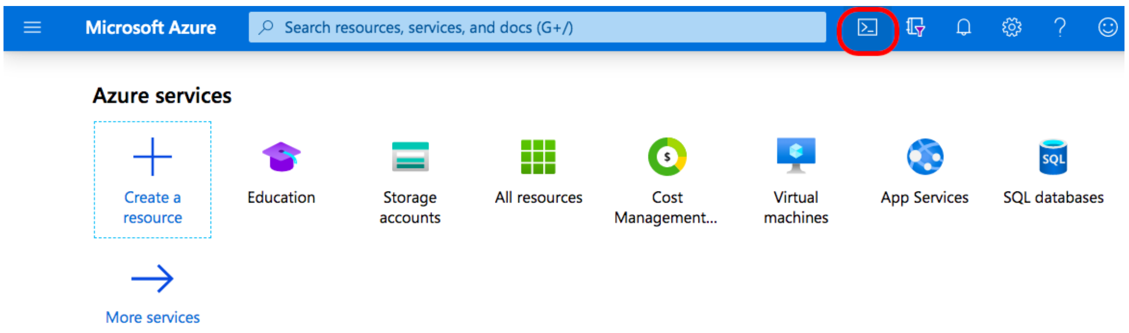
7. In the last step, "Summary", check if the settings are as you intend. These clusters are expensive, so it is worth checking the price estimate at this step: for me it is 1.90 CHF/hour; if your price is larger than this, check your node sizes and counts. When done, initiate the cluster creation by clicking "Create". The process will take time, around 15—25 minutes.

Accessing your cluster

We will interact with the HBase cluster through the command-line interface of HBase. For this, you will need to run the `ssh` program in a terminal in order to connect to your cluster. This process is the same as in week03's HDFS exercise, but we will repeat the instructions here for convenience.

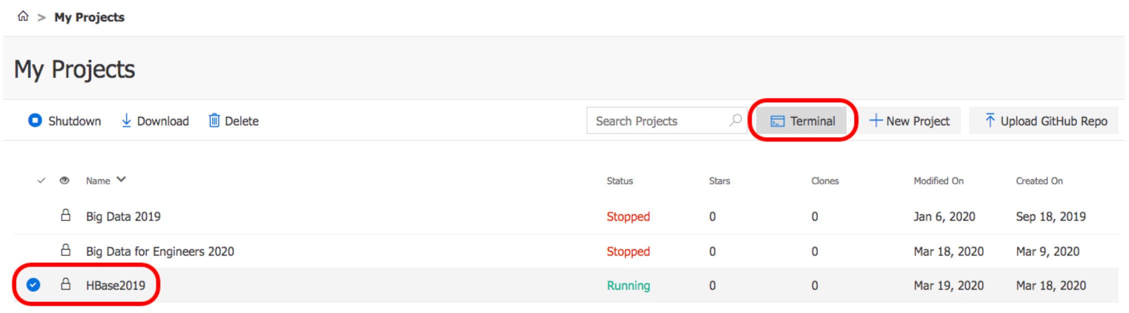
There are three options of how you can do this:

1. **On your own machine** you can just use a normal terminal if you have `ssh` installed. Linux usually has it, as does MacOS. Windows doesn't have it by default (maybe Windows 10 does?), but Windows users can use one of the browser-based options, which are described next, and the other option is to install PuTTY.
2. **In your browser:**
   A. Use the **Azure Cloud Shell**. Click on the Cloud Shell icon at the top of Azure Dashboard toolbar:



   It will ask you to choose between `Bash` and `PowerShell`; select `Bash`. Then it will request your approval for creating a Storage Account required for the shell; agree to it. The shell will then appear at the bottom of the page.

   B. Use a **terminal on Jupyter**. In your notebooks.azure.com tab, click **"My Projects"** at the top of the page. Then, **select one of your projects** and **click "Terminal"**. A shell will be opened in a new page.

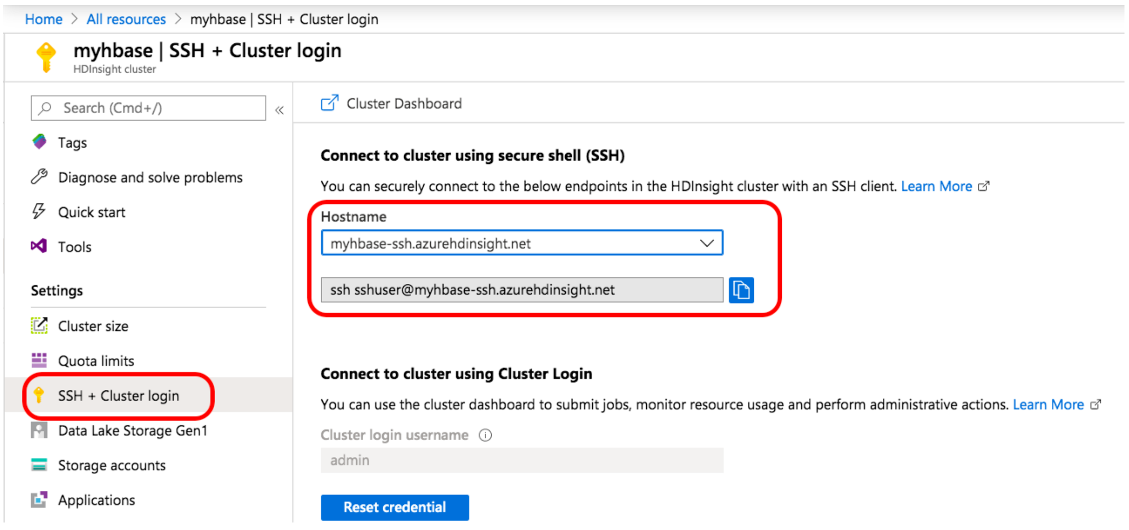


Name	Status	Stars	Clones	Modified On	Created On
Big Data 2019	Stopped	0	0	Jan 6, 2020	Sep 18, 2019
Big Data for Engineers 2020	Stopped	0	0	Mar 18, 2020	Mar 9, 2020
HBase2019	Running	0	0	Mar 19, 2020	Mar 18, 2020

In your terminal of choice, run the following:

```
ssh <ssh_user_name>@<cluster_name>-ssh.azurehdinsight.net
```

In this command, `<ssh_user_name>` is the "ssh username" that you have chosen when creating the HBase cluster, and `<cluster_name>` also comes from that form. Note that the cluster name has to be suffixed with `-ssh`. This command with everything filled-in is also available for copying on **the Azure page of your HBase cluster**, if you click **"SSH + Cluster login"** and **choose the proper hostname**.



Connect to cluster using secure shell (SSH)

You can securely connect to the below endpoints in the HDInsight cluster with an SSH client. Learn More

Hostname: myhbase-ssh.azurehdinsight.net

```
ssh sshuser@myhbase-ssh.azurehdinsight.net
```

Connect to cluster using Cluster Login

You can use the cluster dashboard to submit jobs, monitor resource usage and perform administrative actions. Learn More

Cluster login username: admin

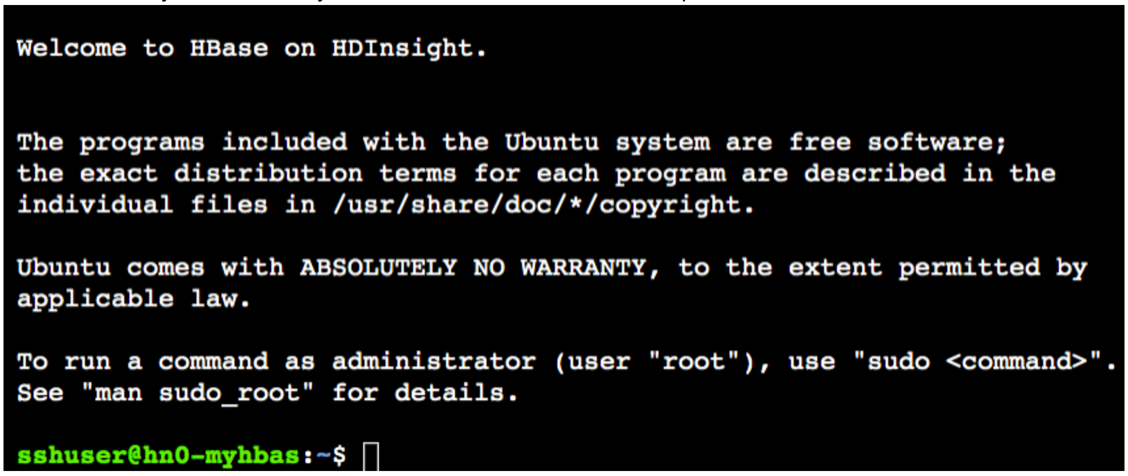
If after running the `ssh` command you see a message similar to this:

```
Welcome to HBase on HDInsight.

Last login: Sat Oct 14 15:56:56 2017 from 180.220.17.157
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

<ssh_user_name>@hn0-cluster:~$
```

then you have successfully connected to your HBase cluster. Here is an example of how it looks like on the Azure cloud terminal:



```
Welcome to HBase on HDInsight.


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

sshuser@hn0-myhbas:~$
```

Troubleshooting

Some issues may arise while creating your HBase cluster. Here are some common issues that we experienced:

1. *StorageAccountAlreadyExists* : Make sure to use a **unique name** while creating a new storage account. The portal does not check for this while in the creation panel but only on validation and an error will arise. This also holds for cluster names.
2. *The ssh connection does not work* : Use the password that you provided at creation. If you can't retrieve it, you can reset the password in the "SSH + Cluster login" panel of your Hbase cluster. Also if you are recreating a new cluster, use a different cluster name as your past created cluster. Otherwise, this may create a conflict in your local *known_hosts* configuration file.

You can find more information about deployement errors on this page.

Interact with your HBase cluster using the shell

In this task we will go through some basic HBase commands, in preparation for the next exercise where we will import a big dataset and run queries against it.

Open the HBase shell

Open the HBase shell by running the following command:

`hbase shell`

Create a table

Let's say we want to create an HBase table that will store sentences adhering to the structure subject-verb-object (e.g., "I eat mangoes", "She writes books") in different languages. Here is a schema that we may use:

Table name = `sentences`

- Column family: `words`
  - column: `subject`
  - column: `verb`
  - column: `object`
- Column family: `info`
  - column: `language`

With the following command we can create such a table (a description of HBase shell commands is available here):

```
create 'sentences', 'words', 'info'
```

You can see the schema of the table with this command:

```
describe 'sentences'
```

Update and query the table

Let's insert some sentences into our table. We will put data cell by cell with the command `put <table>, <rowId>, <columnFamily:columnQualifier>, <value>`:

```
put 'sentences', 'row1', 'words:subject', 'I'
```

```
put 'sentences', 'row1', 'words:verb', 'drink'
```

```
put 'sentences', 'row1', 'words:object', 'coffee'
```

Now, let's try to query this sentence from the table, which we can do with the command `get <table>, <rowId>`:

```
get 'sentences', 'row1'
```

You should see output similar to this:

```
COLUMN                          CELL

 words:object                   timestamp=1555998158489, value=coffee

 words:subject                  timestamp=1555998139704, value=I

 words:verb                     timestamp=1555998148303, value=drink

3 row(s) in 0.0540 seconds
```

As you can see, HBase shell returns data as key-value pairs rather than as rows literally. You may also notice that the lines are **lexicographically sorted** by the key, which is why "subject" appears after "object" in the list.

I don't know how about you, but I like tea more than coffee, so let me update our sentence...

```
put 'sentences', 'row1', 'words:object', 'tea'
```

As you can see, we are using the same `put` command to **update** a cell. But remember that HBase does not actually update cells in place—it just inserts new versions instead. If you now run the query again, you will see the new data:

```
get 'sentences', 'row1'
```

returns:

```
COLUMN                          CELL

 words:object                   timestamp=1555998793452, value=tea

 words:subject                  timestamp=1555998139704, value=I

 words:verb                     timestamp=1555998148303, value=drink

3 row(s) in 0.0470 seconds
```

Scan the table

We actually wanted to store sentences in different languages, so let's first set the language for the existing one:

```
put 'sentences', 'row1', 'info:language', 'English'
```

Note that we are now inserting a value into a different column family but for the same row. Verify with a `get` that this took effect.

Now, let's add a sentence in another language (note that we are using another rowID now— `row2`):

```
put 'sentences', 'row2', 'words:subject', 'Ich'
put 'sentences', 'row2', 'words:subject', 'Ich'
```

```
put 'sentences', 'row2', 'words:verb', 'trinke'
```

```
put 'sentences', 'row2', 'words:object', 'Wasser'
```

```
put 'sentences', 'row2', 'info:language', 'Deutsch'
```

Let's check that we indeed have 2 rows now:

```
count 'sentences'
```

Now, let's query all rows from the table:

```
scan 'sentences'
```

This, indeed, returns all two rows, in key-value format as before.

It is, of course, possible to do some filtering in queries:

- `scan 'sentences', {FILTER => "ValueFilter(=, 'binary:English')"}` will find all cells with the value "English".
- `scan 'sentences', {COLUMNS => 'words:subject', FILTER => "ValueFilter(=, 'substring:I')"}` will find all cells in the column `words:subject` whose value contains a substring "I".
- `scan 'sentences', {COLUMNS => 'words:object', ROWPREFIXFILTER => 'row'}` will find all cells in the column `words:object` whose row key starts with the prefix `row`.

Add a new column to the table

What if we want to store a sentence that also contains an adjective, in addition to the subject, verb, and object? This is not a problem with HBase, because we can create new columns inside **existing** column families on the fly:

```
put 'sentences', 'row3', 'words:subject', 'Grandma'
```

```
put 'sentences', 'row3', 'words:verb', 'bakes'
```

```
put 'sentences', 'row3', 'words:adjective', 'delicious'
```

```
put 'sentences', 'row3', 'words:object', 'cakes'
```

This row now has more columns in the `words` column family than others:

```
get 'sentences', 'row3'
```

We can also add new columns to existing rows:

```
put 'sentences', 'row1', 'words:adjective', 'hot'
```

```
get 'sentences', 'row1'
```

This was a quick overview of HBase shell commands. In the following task we will import a real, sizeable dataset (a subset of Wikipedia) and see how HBase will handle it.

Important: if you do not plan to do the next section right now, please delete your cluster and just recreate it when you need it again.

Exercise 2 — The Wikipedia dataset

Download the dataset

In this task we will see how HBase will handle a large dataset and see how a choice of column families may affect performance.

Let's begin. First, SSH to your cluster as in the previous task:

```
ssh <ssh_user_name>@<cluster_name>-ssh.azurehdinsight.net
```

Note that if you are still in the Hbase shell from the last exercise, you can can quit by entering `exit`.

Download the compressed dataset:

```
wget https://bigdataforeng2020.blob.core.windows.net/exercise05/wikibig.tar.gz
```

Decompress it:

```
tar xvf wikibig.tar.gz
```

The dataset comprises approximately 100,000 articles of the English Wikipedia. You will see four files:

File	What's inside
`text.csv`	Text of the article.
`author.csv`	The username of the latest version's author.
`comment.csv`	Comment that the author left about the last change to the article.
`timestamp.csv`	When that last change was made. Note that this "timestamp" is different from HBase's "timestamp".

The files are in a comma-separated " `key,value` " format in which `key` is the article title.

Upload the dataset to HDFS

Before we can insert the data into HBase, we need to upload it into "HDFS" (for our HDInsight cluster it is actually Azure Blobs). Note that uploading `text.csv` can take a couple of minutes:

```
hdfs dfs -put author.csv /tmp/
```

```
hdfs dfs -put comment.csv /tmp/
```

```
hdfs dfs -put timestamp.csv /tmp/
```

```
hdfs dfs -put text.csv /tmp/
```

Import the dataset to HBase

Let us create the schemas in HBase now

```
hbase shell
```

In order to see what difference column family choice can make, we need to create two different tables, each with a different schema, which we will populate with the same data. One of them will have a single column family (which we name `data`), into which all the four columns (`author` , `timestamp` , `comment` , `text`) will go:

```
create 'wiki_1colfam', 'data'
```

The other table will have two column families—one for **metadata** (`author` , `timestamp` , `comment`) and the other for the article **content** (article text is, of course, larger in size than the metadata):

```
create 'wiki_2colfams', 'metadata', 'content'
```

In both tables, **the row key is the name of the Wikipedia article**.

After the two tables are created, we need to **exit the HBase shell** to return back to the head node's shell:

```
exit
```

Now we need to populate both tables with data. We will use the ImportTsv utility of HBase.

Populate the table `'wiki_1colfam'` by running the following four commands, each of which uploads one column. Note that these commands print a lot of messages, but they are mostly informational with an occasional non-critical warning; unless something goes wrong, of course :) The commands will also report some "Bad Lines", but you can safely ignore this—some lines may contain illegal characters and be dropped, but most of the data is in good shape.

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -
Dimporttsv.columns="HBASE_ROW_KEY, data:author" wiki_1colfam wasbs:///tmp/author.csv
```

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -
Dimporttsv.columns="HBASE_ROW_KEY, data:comment" wiki_1colfam wasbs:///tmp/comment.csv
```

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY, data:timestamp" wiki_1colfam wasbs:///tmp/timestamp.csv
```

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY, data:text" wiki_1colfam wasbs:///tmp/text.csv
```

The last command imports the biggest column, `text`, so it will take time; up to a couple of minutes.

Now we need to populate the other table, `wiki_2colfams`. We will use the same four commands, but notice that we use a different table name and that the `text` column now gets its own column family.

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY, metadata:author" wiki_2colfams wasbs:///tmp/author.csv
```

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY, metadata:comment" wiki_2colfams wasbs:///tmp/comment.csv
```

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY, metadata:timestamp" wiki_2colfams wasbs:///tmp/timestamp.csv
```

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY, content:text" wiki_2colfams wasbs:///tmp/text.csv
```

Tasks to do

1. Write the following queries using the HBase shell:
   A. Print the title and the author's name for each article whose title starts with '`Albert`'.
   B. Print the title and the author's name for each article whose author's name contains the substring '`tom`'.
2. Execute your queries on the two tables (more than once) and observe the query execution times
3. What are the advantages and disadvantages of pure row stores?
4. What are the advantages and disadvantages of pure column stores?
5. What are the advantages and disadvantages of wide column stores?
6. What are the advantages and disadvantages of denormalization?

Solutions

1. The two queries:
   A. All article titles and author names where the article title starts with 'Albert':
      a. `scan 'wiki_1colfam', {COLUMNS => 'data:author', ROWPREFIXFILTER => 'Albert'}`
      b. `scan 'wiki_2colfams', {COLUMNS => 'metadata:author', ROWPREFIXFILTER => 'Albert'}`
   B. All article titles and author names where the author name contains the substring '`tom`'
      a. `scan 'wiki_1colfam', {COLUMNS => 'data:author', FILTER => "ValueFilter(=, 'substring:tom')"}`
      b. `scan 'wiki_2colfams', {COLUMNS => 'metadata:author', FILTER => "ValueFilter(=, 'substring:tom')"}`
2. Execution times
   A. Queries with `ROWPREFIXFILTER` should be quick for both tables, because the filter is applied to the row key rather than to the contents of columns. But even this query could be slower on the table with a single column family, *especially on the first invocation of the query*, because more unrelated data has to be loaded to extract the author name.
   B. The query which searches for a substring in author name takes longer for the table with one column family than for the table with a separate column family for the metadata. HBase stores columns of a single family together and it has to load them together too. So, if for applying a filter to the author column we also have to load the full text of the article (as is the case with just one column family), the operation will take longer than if we don't have to (as is the case with a separate column family for the article text and for metadata). Subsequent invocations of the same command take less time due to caching.
3. **Pure row store:**
   A. Advantages:
      a. Good for workloads with point lookups and updates. Retrieving (updating) a single row is efficient as the row is colocated
   B. Disadvantages:
      a. Scans are more expensive (whole row is always retrieved)
4. **Pure column store:**
   A. Advantages:
      a. Scans are very efficient (only specific columns can be retrieved)
   B. Disadvantages:

    B. Disadvantages:
        a. To retrieve (or update) a whole row, many random accesses need to be performed

5. **Wide column store:**
    A. Advantages:
        a. Column families offer a 'middle ground' between pure row- and column-oriented storages. Columns frequently accessed together can be colocated, very wide columns (affecting scan speed) can be isolated into separate column families
        b. Flexible schema (column names stored for each row) offer flexibility for cases where schema is not known upfront (or in cases of sparse columns)
    B. Disadvantages
        a. Performance penalties, point lookups not as fast as pure row store, scans not as fast as pure column store
        b. The key holds the row key, the column family name, the column qualifier, etc. This can result in a considerable storage overhead if the values that we want to store are small in comparison the key.
6. **Denormalization:**
    A. Advantages:
        a. All operations are either scans or point lookups. No need for expensive joining of multiple relations (all data is colocated or easily mapped)
    B. Disadvantages:
        a. It is difficult to enforce (maintain) consistency in cases of updates
        b. Storage (memory) overhead, due to duplicated data
        c. Scan processing can be more expensive

Important: you may delete your HBase cluster now.

The next exercise will focus on HBase's architecture.

Exercise 3 — Architecture of HBase

In the previous tasks, we have seen HBase in action. Let us now take a look at the internal architecture of HBase. You may want to consult the lecture slides when solving these tasks.

Task 3.1 — Inside a RegionServer

In this exercise you will see how a RegionServer in HBase would execute a query.

Imagine that we have an HBase table called '`phrases`', which has the following schema:

- Column family: `words`
    - column: A
    - column: B
    - column: C
    - (potentially also columns D, E, F, etc.)

Thus, the table has only one column family. Each column in this family holds one word.

Recall from the lecture slides that keys in HBase have the following structure:

row length	row (key)	column family length	column family	column qualifier	timestamp	key type

We need make certain simplifications to the format of keys to avoid excessive clutter in this exercise. Since the table in this exercise has only one column family, we will omit it from the key and will only specify the column name (A,B,C, ...). We will also omit the length fields and the "key type" field. The timestamp field in this exercise will contain integers from 1 to 10, where in reality it would contain the number of milliseconds since an event in the long past. **Thus, keys as will be used in this exercise consist of three fileds: row, column, timestamp.**

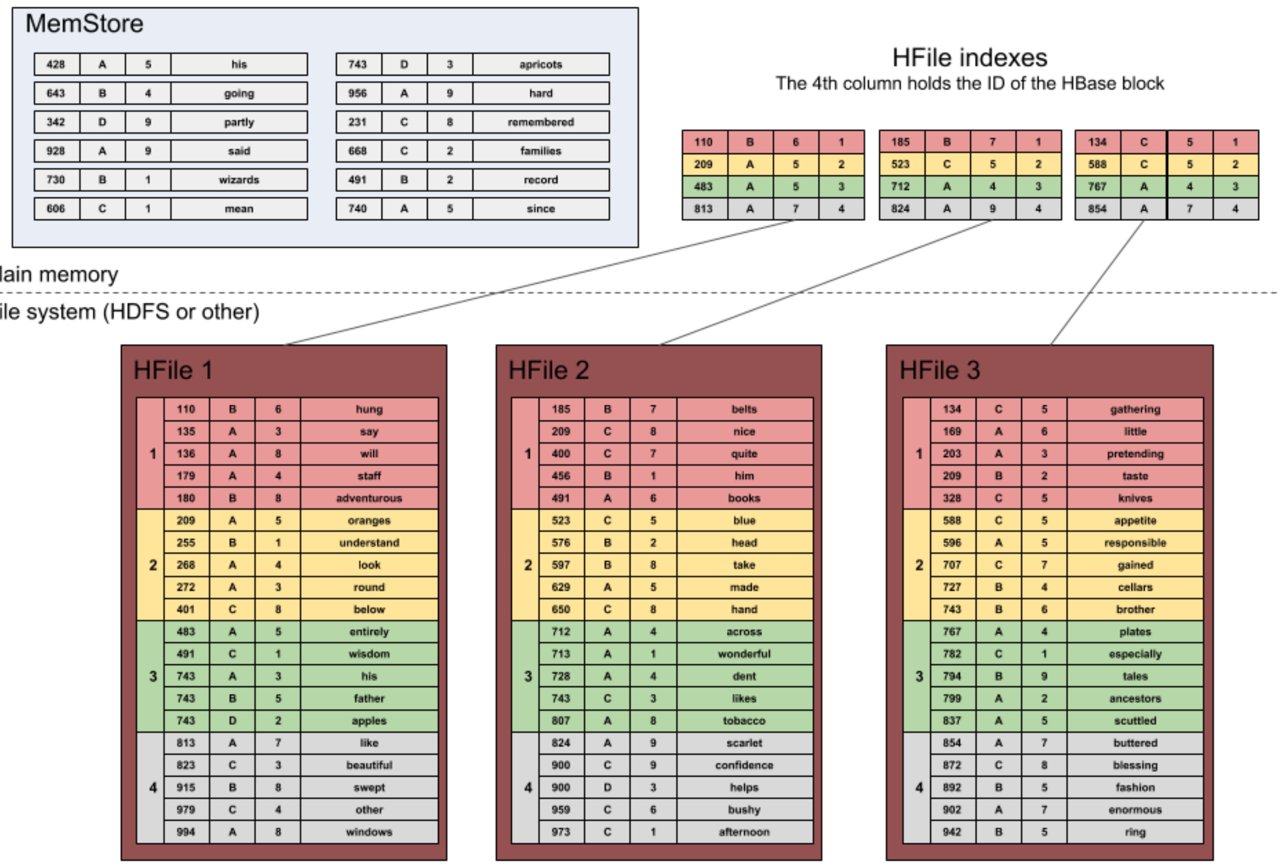
Tasks to do

State which Key-Value pairs will be returned by each of the following queries, given in HBase shell syntax which you have already seen in the first exercise. Assume that the HBase instance is configured to return **only the latest version** of a cell and that the columns are returned in *lexicographic order*.

1. `get 'phrases', '209'`
2. `get 'phrases', '491'`
3. `get 'phrases', '900'`
4. `get 'phrases', '743'`
5. `get 'phrases', '145'`

To answer this question, use the diagram below, which represents the state of a RegionServer responsible for the row region in the range of row IDs 100–999, which is the region into which all these queries happen to fall.

A larger, zoomable, PDF version of this diagram is available here.


MemStore
HFile indexes
The 4th column holds the ID of the HBase block
Main memory
File system (HDFS or other)
HFile 1
HFile 2
HFile 3

MemStore

428	A	5	his
643	B	4	going
342	D	9	partly
928	A	9	said
730	B	1	wizards
606	C	1	mean

743	D	3	apricots
956	A	9	hard
231	C	8	remembered
668	C	2	families
491	B	2	record
740	A	5	since

HFile indexes

The 4th column holds the ID of the HBase block

110	B	6	1
209	A	5	2
483	A	5	3
813	A	7	4

185	B	7	1
523	C	5	2
712	A	4	3
824	A	9	4

134	C	5	1
588	C	5	2
767	A	4	3
854	A	7	4

HFile 1

1	110	B	6	hung
	135	A	3	say
	136	A	8	will
	179	A	4	staff
	180	B	8	adventurous
2	209	A	5	oranges
	255	B	1	understand
	268	A	4	look
	272	A	3	round
	401	C	8	below
3	483	A	5	entirely
	491	C	1	wisdom
	743	A	3	his
	743	B	5	father
	743	D	2	apples
4	813	A	7	like
	823	C	3	beautiful
	915	B	8	swept
	979	C	4	other
	994	A	8	windows

HFile 2

1	185	B	7	belts
	209	C	8	nice
	400	C	7	quite
	456	B	1	him
	491	A	6	books
2	523	C	5	blue
	576	B	2	head
	597	B	8	take
	629	A	5	made
	650	C	8	hand
3	712	A	4	across
	713	A	1	wonderful
	728	A	4	dent
	743	C	3	likes
	807	A	8	tobacco
4	824	A	9	scarlet
	900	C	9	confidence
	900	D	3	helps
	959	C	6	bushy
	973	C	1	afternoon

HFile 3

1	134	C	5	gathering
	169	A	6	little
	203	A	3	pretending
	209	B	2	taste
	328	C	5	knives
2	588	C	5	appetite
	596	A	5	responsible
	707	C	7	gained
	727	B	4	cellars
	743	B	6	brother
3	767	A	4	plates
	782	C	1	especially
	794	B	9	tales
	799	A	2	ancestors
	837	A	5	scuttled
4	854	A	7	buttered
	872	C	8	blessing
	892	B	5	fashion
	902	A	7	enormous
	942	B	5	ring

Solution to the Task 3.1

1. get 'phrases', '209'

Row	Column	Timestamp	Value	Where it came from
209	A	5	oranges	HFile 1
209	B	2	taste	HFile 3
209	C	8	nice	HFile 2

Nothing special here: for that rowID we just return all found key-values. All values come from HFiles.

1. get 'phrases, '491'

Row	Column	Timestamp	Value	Where it came from
491	A	6	books	HFile 2
491	B	2	record	MemStore
491	C	1	wisdom	HFile 1

In this case, one value came from the MemStore.

1. get 'phrases', '900'

Row	Column	Timestamp	Value	Where it came from
900	C	9	confidence	HFile 2
900	D	3	helps	HFile 2

This example lacks values for columns "A" and "B", and this is perfectly fine with HBase. Also, both values come from the same HFile.

1. get 'phrases', '743'

Row	Column	Timestamp	Value	Where it came from
743	A	3	his	HFile 1
743	B	6	brother	HFile 3
743	C	3	likes	HFile 2
743	D	3	apricots	MemStore

In this example, versions matter. The value `743,B,5,father` from HFile 1 gets superceded by `743,B,6,brother` from HFile 3, which has a higher version number. The same happens to value `743,D,2,apples` from HFile 1, which gets superceded by `743,D,3,apricots` from MemStore.

1. get 'phrases', '145'

The rowID `145` does not exist, so nothing is returned.

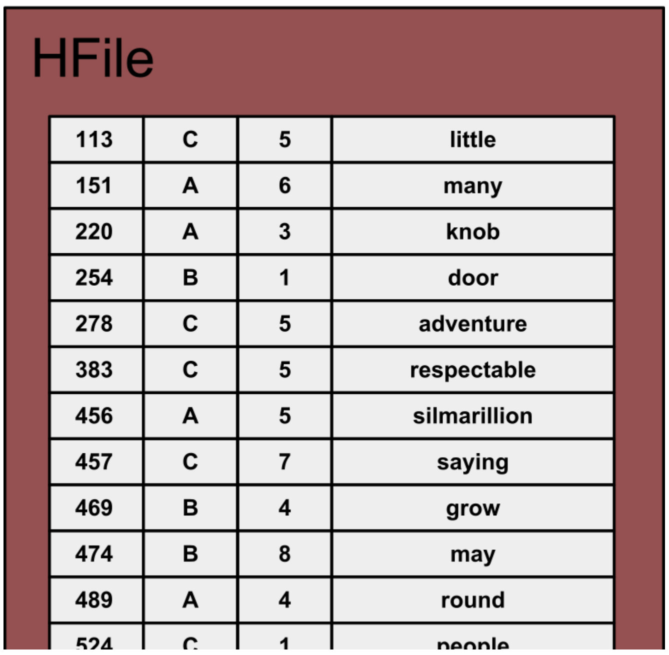
Own exploration

Building an HFile index

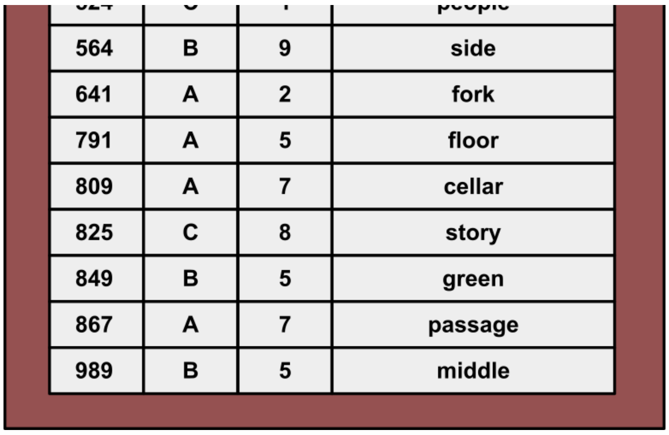
When performing a get, the RegionServer needs to check its MemStore and all HFiles (unless the Bloom filter returns negative) for the existence of the requested key. In order to avoid scanning HFiles entirely, HBase uses index structures to quickly skip to the position of the *HBase block* which may hold the requested key.

By default, each *HBase block* is 64KB (configurable) in size and always contains whole key-value pairs, so, if a block needs more than 64KB to avoid splitting a key-value pair, it will just grow.

In this task, you will be building the index of an HFile. **For the purpose of this exercise**, assume that each HBase block is 40 bytes long, and each character in keys and values is worth 1 byte: for example, the first key-value pair in the diagram below is worth $3 + 1 + 1$ bytes. Below this diagram you will find a table for you to fill in.
$+ 6 = 11$


HFile

113	C	5	little
151	A	6	many
220	A	3	knob
254	B	1	door
278	C	5	adventure
383	C	5	respectable
456	A	5	silmarillion
457	C	7	saying
469	B	4	grow
474	B	8	may
489	A	4	round
524	C	1	people

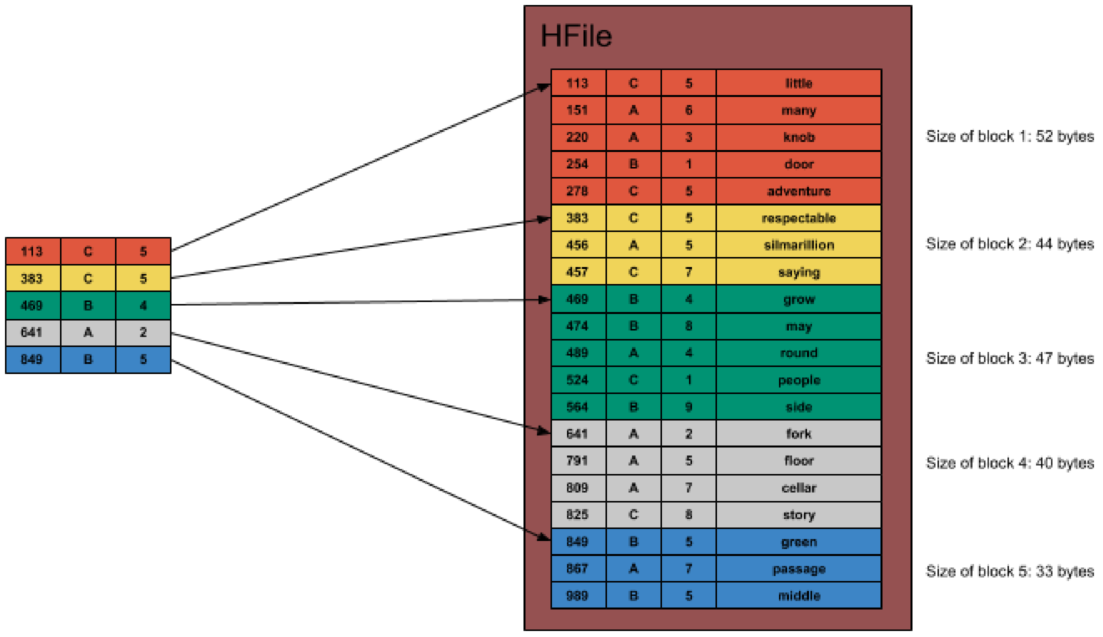


564	B	9	side
641	A	2	fork
791	A	5	floor
809	A	7	cellar
825	C	8	story
849	B	5	green
867	A	7	passage
989	B	5	middle

Based on the contents of the HFile above, you need to **populate the index**, following the approach described in the lecture slides. Use the following table (again, you can edit it by double-clicking). Use as many or as few rows as you need.

RowId	Column	Version

Solution



113	C	5
383	C	5
469	B	4
641	A	2
849	B	5

113	C	5	little
151	A	6	many
220	A	3	knob
254	B	1	door
278	C	5	adventure
383	C	5	respectable
456	A	5	silmarillion
457	C	7	saying
469	B	4	grow
474	B	8	may
489	A	4	round
524	C	1	people
564	B	9	side
641	A	2	fork
791	A	5	floor
809	A	7	cellar
825	C	8	story
849	B	5	green
867	A	7	passage
989	B	5	middle

Size of block 1: 52 bytes

Size of block 2: 44 bytes

Size of block 3: 47 bytes

Size of block 4: 40 bytes

Size of block 5: 33 bytes

```
In [ ]:
```