Show Submission Credentials

P3. HBase Basics An introduction to HBase

16 days 3 hours left

Introduction to HBase

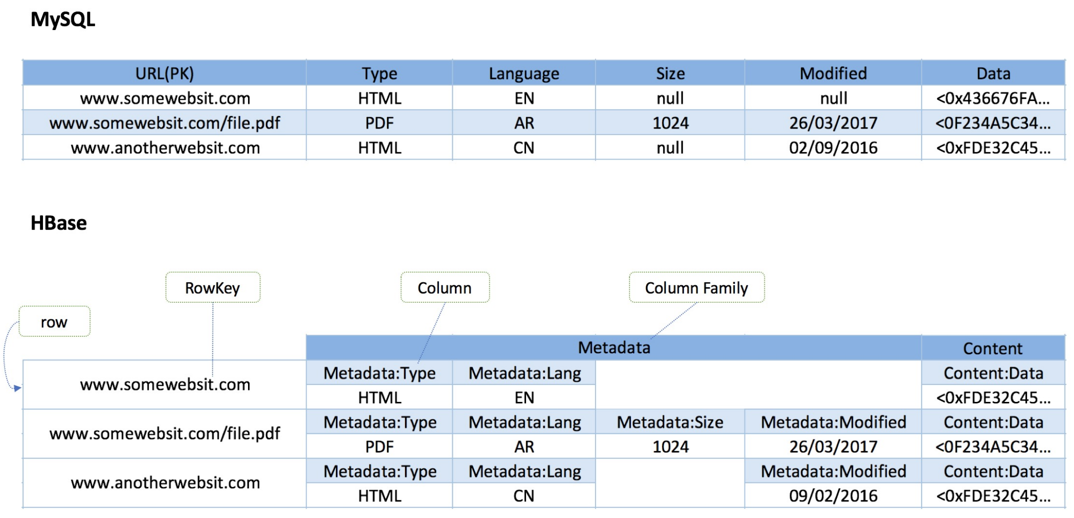
Introduction to HBase

Apache HBase is an open-source version of Google's BigTable (https://en.wikipedia.org/wiki/Bigtable) distributed storage system supported by the Apache Software Foundation. BigTable is a sparse, distributed, scalable, high-performance, versioned database. BigTable's infrastructure is designed to store billions of rows and columns of data in loosely defined tables. Just as traditional RDBMSs are designed to run on top of a local filesystem, HBase is designed to work on top of the Hadoop Distributed File System (HDFS) (https://en.wikipedia.org/wiki/Apache_Hadoop#Hadoop_distributed_file_system). HDFS is a distributed file system that stores files as replicated blocks across multiple servers.

Sparse feature in HBase

HBase table is **physically stored by column family** sparsely so that rows in the same table can have heavily-varying columns. Cells in the table from the conceptual view that appear to be empty **do not take space or physically exist**. This is what makes HBase "sparse".

The following figure addresses the difference in how HBase data is physically organized compared to RDBMS:



MySQL

URL(PK)	Type	Language	Size	Modified	Data
www.somewebsit.com	HTML	EN	null	null	<0x436676FA...
www.somewebsit.com/file.pdf	PDF	AR	1024	26/03/2017	<0F234A5C34...
www.anotherwebsit.com	HTML	CN	null	02/09/2016	<0xFDE32C45...

HBase

	Metadata				Content
www.somewebsit.com	Metadata:Type	Metadata:Lang			Content:Data
	HTML	EN			<0xFDE32C45...
www.somewebsit.com/file.pdf	Metadata:Type	Metadata:Lang	Metadata:Size	Metadata:Modified	Content:Data
	PDF	AR	1024	26/03/2017	<0F234A5C34...
www.anotherwebsit.com	Metadata:Type	Metadata:Lang		Metadata:Modified	Content:Data
	HTML	CN		09/02/2016	<0xFDE32C45...

Figure 1: An example of organized data in MySQL and HBase

A row in HBase is referenced using a rowkey which is a **raw byte array**, which can be considered to be the primary key of the table in an RDBMS. The primary key of the table has to be unique and hence is mapping to one and only one row. HBase automatically sorts table rows by rowkey when it stores them. By default, this sort is byte ordered.

Each column in HBase has a column name. Columns can be further grouped into column families. All columns belonging to the same column family share the same prefix. For example, in the figure above, the columns `Metadata:Type` and `Metadata:Lang` are both members of the `Metadata` column family, and the column `Content:Data` is a member of the `Content` column family. By default, the colon character `:` delimits the column prefix from the family member. The column family prefixes must be composed of **printable characters**. The qualifying tail can contain any arbitrary bytes. **Note that although column families are fixed at table creation, column qualifiers are mutable and may differ between rows.**

A **cell** is a combination of **row, column family, and column qualifier, and at least one version(s).** Each version has its own value and timestamp. Thus A {row, column_family:column, version} tuple exactly specifies a version of a cell in HBase. Similar to Rowkey, value is simply bytes, so any value that could be serialized into bytes can be stored in a cell. Rows in HBase are sorted lexicographically by rowkey which is a very important property as it allows for quick searching. You have learned the definition of lexicographical order in Project 1 and it will help you in this project. You may notice that the **timestamp** isn't explicitly noted in the above diagram. We are assuming that the sample table here has the attribute `VERSIONS => '1'`. The default value of `VERSIONS` given a column qualifier will be 1 if not explicitly set. Setting maximum-versions to 1 means each put still creates a new version but only the latest one is kept. In this case, a {row, column_family:column} tuple can be adequate to specify a value stored in HBase. However, you should understand the behavior of the HBase version dimension, which is vital to understanding the behaviors of the core HBase operations.

Take `Get` operation as an example:

```
# By default, if you specify no explicit version, when doing a get, the cell whose ver
sion has the largest timestamp is returned
get 'table', 'row', { COLUMN => 'column' }
# to return more than one version
get 'table', 'row', { COLUMN => 'column',  VERSIONS => 5 }
# to get the value by a specified timestamp
get 'table', 'row', { COLUMN => 'column', TIMESTAMP => 1317945301466 }
# or by a range of timestamps, note that this will ONLY return the version with the la
rgest timestamp in this range
get 'table', 'row', { COLUMN => 'column', TIMERANGE => [first_timestamp, second_timest
amp] }
# to get multiple values in a range of timestamps, you need to set VERSIONS explicitly
get 'table', 'row', { COLUMN => 'column', TIMERANGE => [first_timestamp, second_timest
amp], VERSIONS => 5 }
```

HBase Operations

HBase has four primary operations on the data model: **Get**, **Put**, **Scan**, and **Delete**.

A `Get` operation returns all of the cells for a specified row identified by a rowkey. You can further narrow the scope of what to `Get` by specifying column families, columns, timestamps, versions, and filters.

`Scan` is an operation that iterates over multiple rows for specific attributes based on some conditions. Similarly, you can limit the scope by specifying column families, columns, timestamps, versions, and filters. When using HBase Java APIs, if you try to limit the maximum number of values returned for each call to `ResultScanner#next()`, call `Scan#setBatch(int)`.

Get and Scan operations always return data in sorted order. Data are first sorted by rowkeys, then by column family, then by column qualifier, and finally by timestamp (so the values with largest timestamps appear first).

A `Put` operation can either add new rows to the table when used with a new key or update a row if the key already exists. It always creates a new version of a cell at a certain timestamp. By default the system uses the server's `System#currentTimeMillis()`, but you can specify the version (i.e. a long integer) yourself on a per-column level. This means it is possible to assign a time in the past or the future. To overwrite an existing value, do a `put` at the same row, column, and version to overwrite.

A `Delete` operation removes value(s) from a table in one of the following levels: a specified version of a column, all versions of a column, all columns of a column family, or the entire row. Note that HBase does not modify data in place. `Delete` will just create a new tombstone marker for the row to delete. These tombstones will be removed during compaction.

You need to understand all the core operations can manipulate different levels of data, not just by rows. Nevertheless, by default, `Get` and `Scan` operations on an HBase table are performed on data whose version has the largest timestamp. A `Put` operation always creates a new version of the data that are put into HBase and `Delete` operations delete an entire row.

HBase Architecture

HBase is composed of three types of servers in a master/slave type of architecture. `HRegionServer` serves data for read and writes. When accessing data, clients directly communicate with `HRegionServer`. Region assignment, DDL(create, delete, etc) operations are handled by the `HMaster`. `Zookeeper`, which is a distributed coordination service, maintains a live cluster state. All HBase data is stored in `HFiles`. The basic architecture of HBase is shown below:

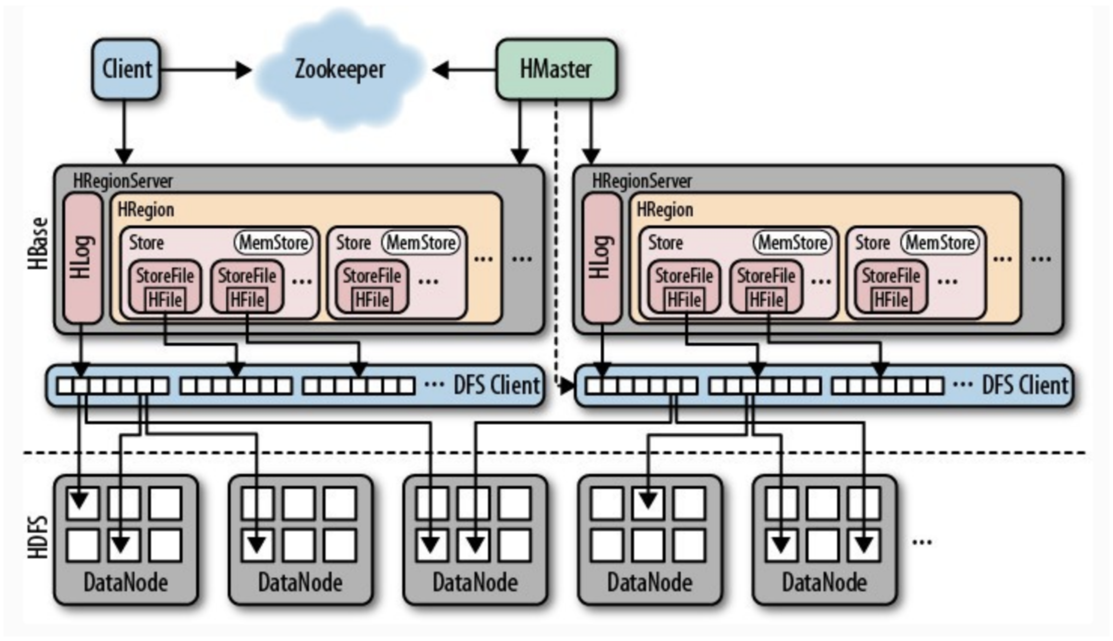


Figure 2: HBase cluster structure

Catalog Tables: The catalog table `hbase:meta` exists as an HBase table that keeps a list of all regions in the system, and the location of `hbase:meta` is stored in Zookeeper.

Client: The HBase client finds the HRegionServers that serving the particular row range of interest. It does this by querying the `hbase:meta` table. After locating the required region, the client contacts the HRegionServer serving that region, rather than going through the HMaster. This information is cached in the client so that subsequent requests need not go through the lookup process.

Zookeeper: Maintain the server status via heartbeat. By registering each node in the cluster to Zookeeper, HMaster can sense the health status of each HRegionServer.

HMaster: Manage and monitor the health of all HRegionServers. When a new HRegionServer connects to HMaster, HMaster tells it to wait for data to be allocated. When an HRegionServer dies, HMaster marks all HRegions it is responsible for as unallocated and then assigns them to other HRegionServers. To avoid single point failure. HBase can start multiple HMasters and only keep one HMaster running through the Zookeeper's election mechanism.

HRegion: When the size of the table exceeds the preset value, HBase will automatically divide the table into different regions, each of which contains a contiguous subset of all the rows in the table. HRegion is composed of multiple HStores. Each HStore corresponds to the storage of one column family in the logical table. Therefore, to improve operational efficiency, it is preferable to place columns with common I/O characteristics in one column family.

HRegionServer: Be mainly responsible for reading and writing data to HDFS file system in response to user I/O requests. Each HRegionServer can serve several HRegions, but one HRegion can only be served by a specific HRegionServer.

HStore: Consist of MemStore and StoreFiles. MemStore is a write cache, which stores updates in memory as sorted KeyValues. The data written by the user will first be put into MemStore. When MemStore is full, the data is MemStore is flushed into a new StoreFile (the underlying implementation is HFile). When the number of StoreFile files increases to a certain threshold, the major compaction will be triggered, which is introduced in the later sections.

HLog: Implement the WAL (Write Ahead Log). Each time when the client wants to write data to MemStore, it needs to write a copy of the data to the HLog first. Edits are appended to the end of the HLog that is stored on disk. HLog periodically deletes old data, which has been persisted to the StoreFile. When HRegionServer crashes, the HLog is used to recover not-yet-persisted data.

HFile: An HFile contains a set of sorted key-value pairs. But in fact, an HFile is divided into blocks to make the jobs of splitting, reading, caching, indexing and compression easier. There are blocks for storing index, metadata, Bloom Filter and data. Each data block is 64KB large by default and contains a magic header and a number of serialized KeyValue instances. Each KeyValue instances is just a byte array. You can see its format in the following picture.

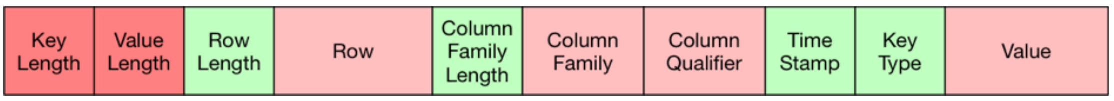


Figure 3: The HFile structure (Reference: HBase: The Definitive Guide, 1st edition, page 329)

BlockCache: This is a read cache and stores frequently read data in memory. Least Recently Used data is evicted when full. Each HRegionServer only has a single BlockCache instance. HBase provides three options for BlockCache: **LruBlockCache**, **SlabCache**, **BucketCache**. You can refer to this blog (https://blog.cloudera.com/hbase-blockcache-101/) to see their differences.

DataNode: All HBase data is stored in HDFS DataNodes. RegionServers are collocated with the HDFS DataNodes, which enable data locality (putting the data close to where it is needed) for the data served by the RegionServers. HBase data is local when it is written since an HDFS client writes the first replica of data to local nodes, but when a region is moved, it is not local until data compaction. For more information about HBase locality, see this link (https://hbase.apache.org/book.html#regions.arch.locality).

Information

Notes

1. Even though an HBase Region can only be served by one RegionServer, this does not mean the data of that Region can only exist in one RegionServer. In fact, due to data replication of HDFS, there will be some extra copies of the data of each Region on another RegionServer.
2. Here are our references. We highly suggest you read through some of these to gain deeper and more detailed knowledge of how HBase works.
    - HBase Working Principle: A part Hadoop Architecture (https://towardsdatascience.com/hbase-working-principle-a-part-of-hadoop-architecture-fbe0453a031b)
    - HBase Reference Guide (http://hbase.apache.org/book.html)
    - Apache HBase I/O – HFile (http://blog.cloudera.com/blog/2012/06/hbase-io-hfile-input-output/)
    - Apache HBase Write Path (http://blog.cloudera.com/blog/2012/06/hbase-write-path/)
    - Apache HBase Region Splitting and Merging (https://hortonworks.com/blog/apache-hbase-region-splitting-and-merging/)
    - HBase: The Definitive Guide (http://shop.oreilly.com/product/0636920014348.do)

HBase Tutorial

HBase Tutorial

Set up Standalone HBase

This section describes the setup of a standalone HBase. A standalone instance has all HBase daemons, the Master, RegionServers, and ZooKeeper, running in a single JVM persisting to the local filesystem.

1. HBase requires that a JDK be installed. We will use HBase 2.3.6 and JDK 8 in this primer.
2. Download the HBase file `hbase-2.3.6-bin.tar.gz` from this link (https://downloads.apache.org/hbase/2.3.6/hbase-2.3.6-bin.tar.gz).
3. Extract the downloaded file and change it to the newly-created directory.

```
tar xzvf hbase-2.3.6-bin.tar.gz
cd hbase-2.3.6
```

4. You must set the `JAVA_HOME` environment variable before starting HBase. You can use the command `whereis java` to locate the installed java. Once you have the location, edit the `conf/hbase-env.sh` file, uncomment the line starting with `#export JAVA_HOME=` , and then set it to your java installation path. Below is an example:

```
export JAVA_HOME=/usr/jdk64/jdk1.8.0_112
```

5. Edit `conf/hbase-site.xml` , which is the main HBase configuration file. You need to specify the directory on the local filesystem where HBase and ZooKeeper write data and acknowledge some risks. Below is an example `hbase-site.xml` for standalone HBase, replace the value for `hbase.rootdir` and `hbase.zookeeper.property.dataDir` with your directory.

```
<configuration>
    <property>
        <name>hbase.rootdir</name>
        <value>file:///home/testuser/hbase</value>
    </property>
    <property>
        <name>hbase.zookeeper.property.dataDir</name>
        <value>/home/testuser/zookeeper</value>
    </property>
    <property>
        <name>hbase.unsafe.stream.capability.enforce</name>
        <value>false</value>
        <description>
        Controls whether HBase will check for stream capabilities (hflush/hsync).

        Disable this if you intend to run on LocalFileSystem, denoted by a rootdi
r
        with the 'file://' scheme, but be mindful of the NOTE below.

        WARNING: Setting this to false blinds you to potential data loss and
        inconsistent system state in the event of process and/or node failures. I
f
        HBase is complaining of an inability to use hsync or hflush it's most
        likely not a false positive.
        </description>
    </property>
</configuration>
```

6. Start HBase by running `start-hbase.sh` in the `bin` directory

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

7. Connect to your running HBase using the `hbase shell` command, located in the `bin` directory of your installed HBase. After that, you can practice HBase shell commands locally.

```
./bin/hbase shell
```

8. To stop the local HBase, you can run the script `stop-hbase.sh` in the `bin` directory.

```
./bin/stop-hbase.sh
```

Information

Notes

1. If you are using macOS, `whereis java` might give you a symbolic link. Instead, you may use the command `$(dirname $(readlink $(which javac)))/java_home`. However, it's recommended to use Linux for this task.
2. You do not need to create the HBase data directory. HBase will do this for you. If you create the directory, HBase will attempt to do a migration, which is not what you want.
3. You can use the `jps` command to see whether HBase is running.
4. For more information about standalone HBase and pseudo-distributed HBase, please refer to this reference guide (https://hbase.apache.org/book.html#quickstart) from Apache HBase.

Danger

Please remember that EMR and HDInsight can be expensive, you may want to use spot instances to reduce spending on EMR and to control the number or the size of nodes of your HDInsight clusters.

Warning: your Azure subscription will be disabled if you run out of the budget in your subscription. You may submit a request to the course staff to reactivate your subscription and increase the budget, however, you should be mindful of your spending.

The following sections introduce how to provision HBase clusters on AWS and Azure.

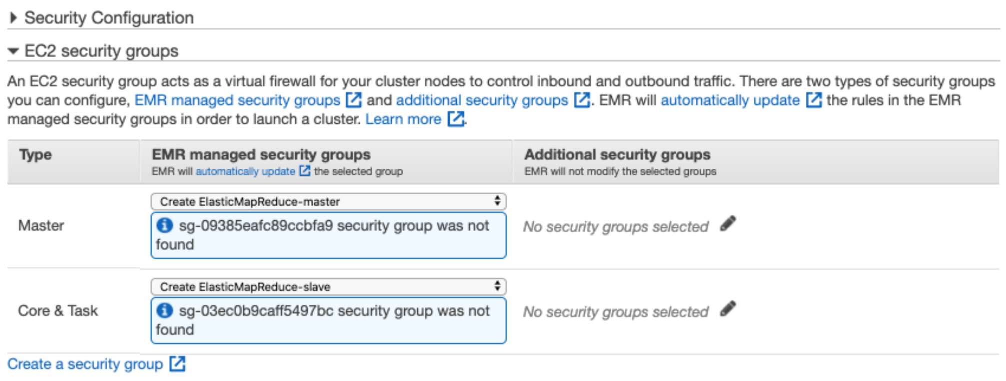
Creating the HBase Cluster with AWS EMR

One solution to provision an HBase cluster is to use Amazon's Elastic MapReduce (EMR). HBase uses the Hadoop Distributed File System (HDFS) to store data. HDFS is a file system that is distributed over the individual core nodes in an EMR cluster. By default, Amazon configures HDFS in an EMR cluster to use the built-in instance storage volumes that come by default in each core EC2 instance.

The subsequent steps will illustrate how you can run an EMR cluster to work with HBase.

1. Select "Go to advanced options" on the top of the "create cluster" page
2. Choose **Release: EMR-5.29.0** as the release version.
3. Remove unrelated additional services such as Pig, Hue and Hive and choose to install **Hadoop 2.8.5 and HBase 1.4.10** only. Select HDFS as the storage mode.
4. Launch an EMR cluster with **1 master** and **2 core** nodes. Make sure all the instances are `m4.large` and that you've requested spot instances with an appropriate bid price. To keep on top of optimal pricing, check AWS's EC2 Spot Instance Pricing here (https://aws.amazon.com/ec2/spot/pricing/).

5. Make sure that the EMR cluster is created within the same VPC as your project submitter instance (the one with your runner.sh) if you are working on projects.
6. Do not forget to tag your EMR cluster with Key: `project` and Value as the tag value being used in the **currently ongoing project**, which will be automatically propagated to the individual EC2 instances in the cluster in most cases.
7. You must specify an EC2 key pair to be able to SSH to the individual instances.
8. Set the security group for both **Master** and **Slave** nodes as described below. If **ElasticMapReduce-master** and **ElasticMapReduce-slave** do not exist, create them.



▸ Security Configuration

▾ EC2 security groups

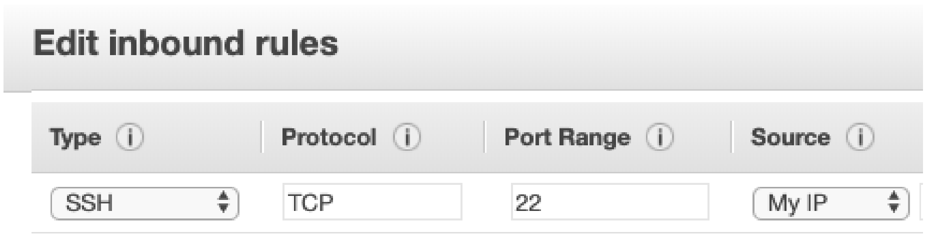
An EC2 security group acts as a virtual firewall for your cluster nodes to control inbound and outbound traffic. There are two types of security groups you can configure, EMR managed security groups and additional security groups. EMR will automatically update the rules in the EMR managed security groups in order to launch a cluster. Learn more

Type	EMR managed security groups EMR will automatically update the selected group	Additional security groups EMR will not modify the selected groups
Master	Create ElasticMapReduce-master sg-09385eafc89ccbfa9 security group was not found	*No security groups selected*
Core & Task	Create ElasticMapReduce-slave sg-03ec0b9caff5497bc security group was not found	*No security groups selected*

Create a security group

Figure 4: EMR Security Group

9. After creating the EMR cluster, remember to open the **SSH port** for **My IP** in the security group **ElasticMapReduce-master**. So that you can connect to the EMR cluster via SSH.



Edit inbound rules

Type	Protocol	Port Range	Source
SSH	TCP	22	My IP

Figure 5: An example of opening an SSH port

Danger

You should not open all the ports of the master and core nodes to the public as this will expose the vulnerability of DDOS attacks that could block you from using the HBase clusters and result in a high data transfer cost.

Use the public DNS of the master node as the SSH DNS. Note that the username to use while SSHing to these instances is `hadoop`. After you SSH into master node, you can run the following command to verify that HDFS is healthy given how it's being reported per data node:

```
hdfs dfsadmin -report
```

You can then provision an EC2 instance as the client machine and connect the instance to the HBase cluster.

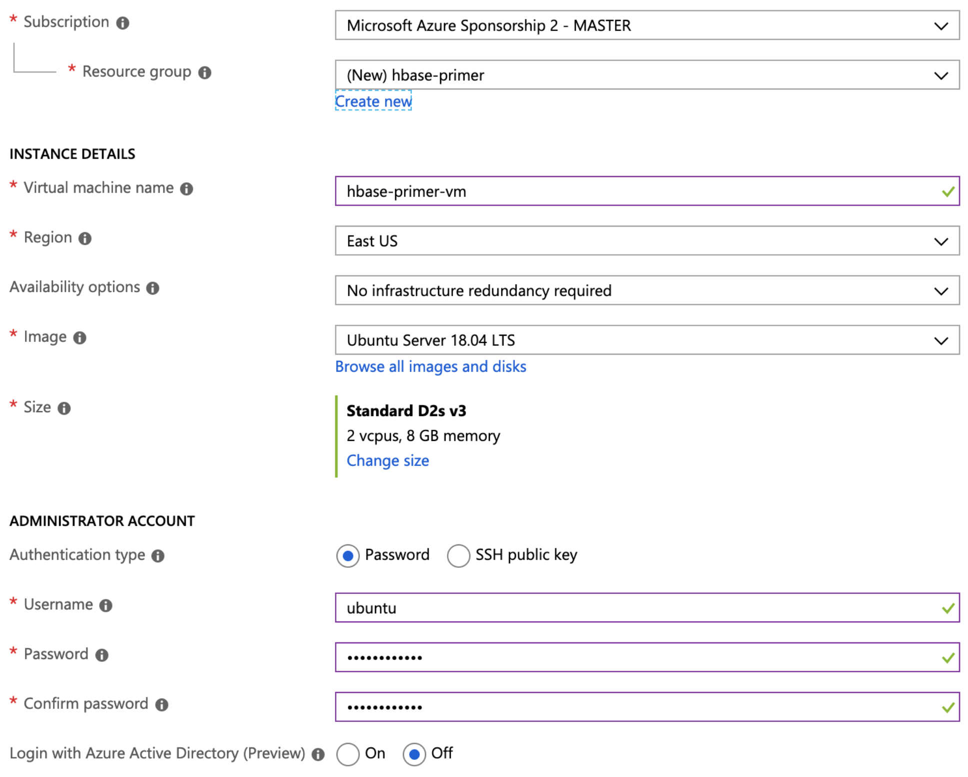
Creating the HBase Cluster with Azure HDInsight

Alternatively, you can provision an HBase cluster on Azure HDInsight.

This section describes how you provision an Azure VM as the client machine and an HBase cluster on Azure HDInsight.

First, provision an Azure VM which you will use in the HBase Java API section.

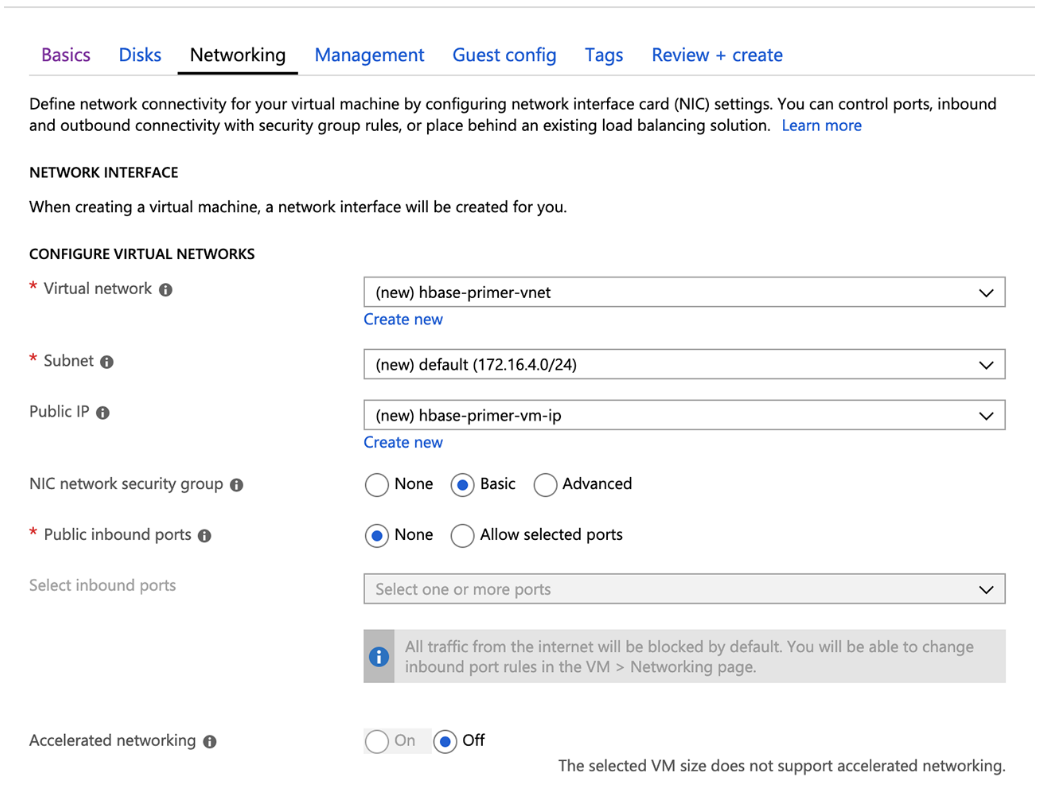
1. On the Azure portal, create an Ubuntu Server 18.04 VM.
2. In the "Basics" tab, create a new resource group, for example, a resource group named "hbase-primer". You may select Password as the Authentication type..



* Subscription	Microsoft Azure Sponsorship 2 - MASTER
* Resource group	(New) hbase-primer
INSTANCE DETAILS	
* Virtual machine name	hbase-primer-vm
* Region	East US
Availability options	No infrastructure redundancy required
* Image	Ubuntu Server 18.04 LTS
* Size	Standard D2s v3 2 vcpus, 8 GB memory
ADMINISTRATOR ACCOUNT	
Authentication type	[✓ Password] SSH public key
* Username	ubuntu
* Password	••••••••••••
* Confirm password	••••••••••••
Login with Azure Active Directory (Preview)	On [✓ Off]

Figure 6: An example of the basic configuration for VM

3. Keep the default setting at the "Disks" tab. In the "Network" tab, create a new Virtual network in the resource group. Note down the Virtual Network ID. In the figure below, for example, the ID is `hbase-primer-vnet` .



Define network connectivity for your virtual machine by configuring network interface card (NIC) settings. You can control ports, inbound and outbound connectivity with security group rules, or place behind an existing load balancing solution. Learn more

NETWORK INTERFACE

When creating a virtual machine, a network interface will be created for you.

CONFIGURE VIRTUAL NETWORKS

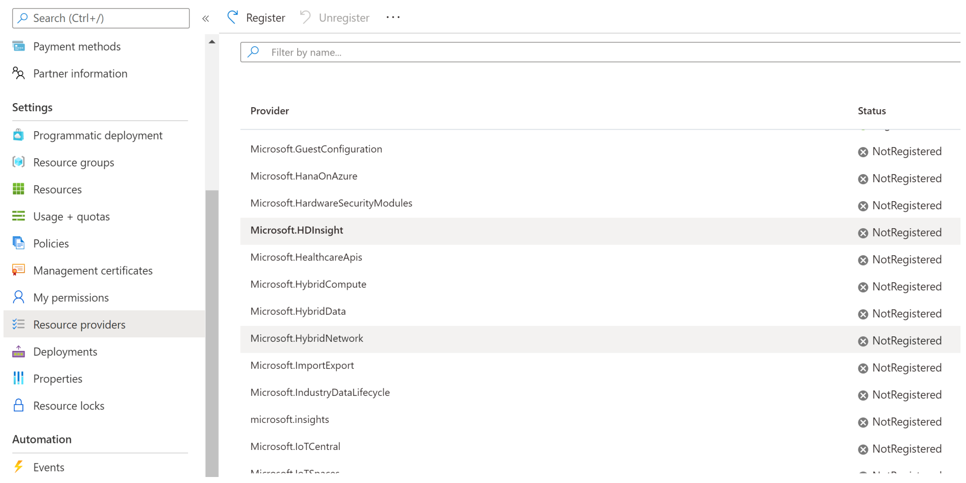
* Virtual network	(new) hbase-primer-vnet Create new
* Subnet	(new) default (172.16.4.0/24)
Public IP	(new) hbase-primer-vm-ip Create new
NIC network security group	None [✓ Basic] Advanced
* Public inbound ports	[✓ None] Allow selected ports
Select inbound ports	Select one or more ports
	All traffic from the internet will be blocked by default. You will be able to change inbound port rules in the VM > Networking page.
Accelerated networking	On [✓ Off] The selected VM size does not support accelerated networking.

Figure 7: An example of the network configuration for VM

4. Do not forget to tag your resources with Key: `project` and Value as the tag value being used in the **currently ongoing project**.
5. Navigate to the "Review + create" tab to provision the VM.

Second, provision an HDInsight cluster in the same virtual network as the VM.

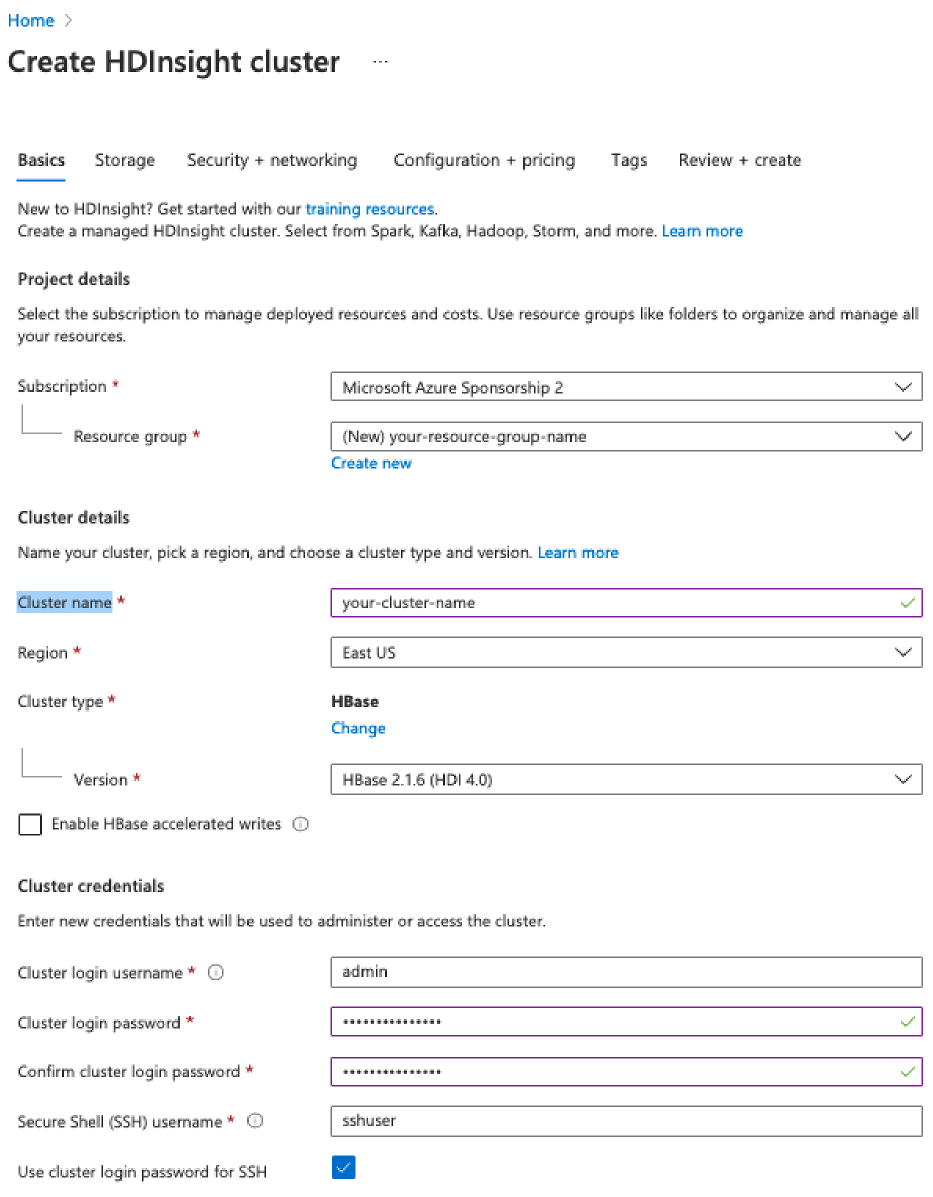
1. If it is the first time for your subscription to create a HDInsight cluster, you may want to check out the service providers for CPU quota limits of HDInsight.
    1. Go to the subscription page of Azure portal, click the subscription "Microsoft Azure Sponsorship 2", and select **Resource Providers** from the left panel.
    2. Select **Microsoft.HDInsight** from the right panel. If not registered, click on the register button at the left top of the panel to register it.
    3. You can expect to have 40 quota limits on HDInsight in East-US within 5 minutes.



Provider	Status
Microsoft.GuestConfiguration	NotRegistered
Microsoft.HanaOnAzure	NotRegistered
Microsoft.HardwareSecurityModules	NotRegistered
Microsoft.HDInsight	NotRegistered
Microsoft.HealthcareApis	NotRegistered
Microsoft.HybridCompute	NotRegistered
Microsoft.HybridData	NotRegistered
Microsoft.HybridNetwork	NotRegistered
Microsoft.ImportExport	NotRegistered
Microsoft.IndustryDataLifecycle	NotRegistered
microsoft.insights	NotRegistered
Microsoft.IoTCentral	NotRegistered

Figure 8: Register Microsoft.HDInsight as a resource provider.

2. Click "Create a resource" on the sidebar of the Azure portal. Search "Azure HDInsight" in the search box and click "Create".
3. In the "Basics" step, input the following information:
    1. A global unique cluster name
    2. Cluster type: HBase, HBase 1.1.2
    3. Location: East US
    4. Select the same Resource group as the VM
    5. Note the difference between the Cluster login username and SSH username. "Cluster login username" is used when you access the dashboard from the browser, and SSH username is for SSH connections. Cluster login username and SSH username may share the same password but the usernames must be different from each other.



New to HDInsight? Get started with our training resources.
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 *	Microsoft Azure Sponsorship 2
Resource group *	(New) your-resource-group-name

Create new

Cluster details

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

Cluster name *	your-cluster-name
Region *	East US
Cluster type *	HBase
Version *	HBase 2.1.6 (HDI 4.0)

Change

Enable HBase accelerated writes [ ]

Cluster credentials

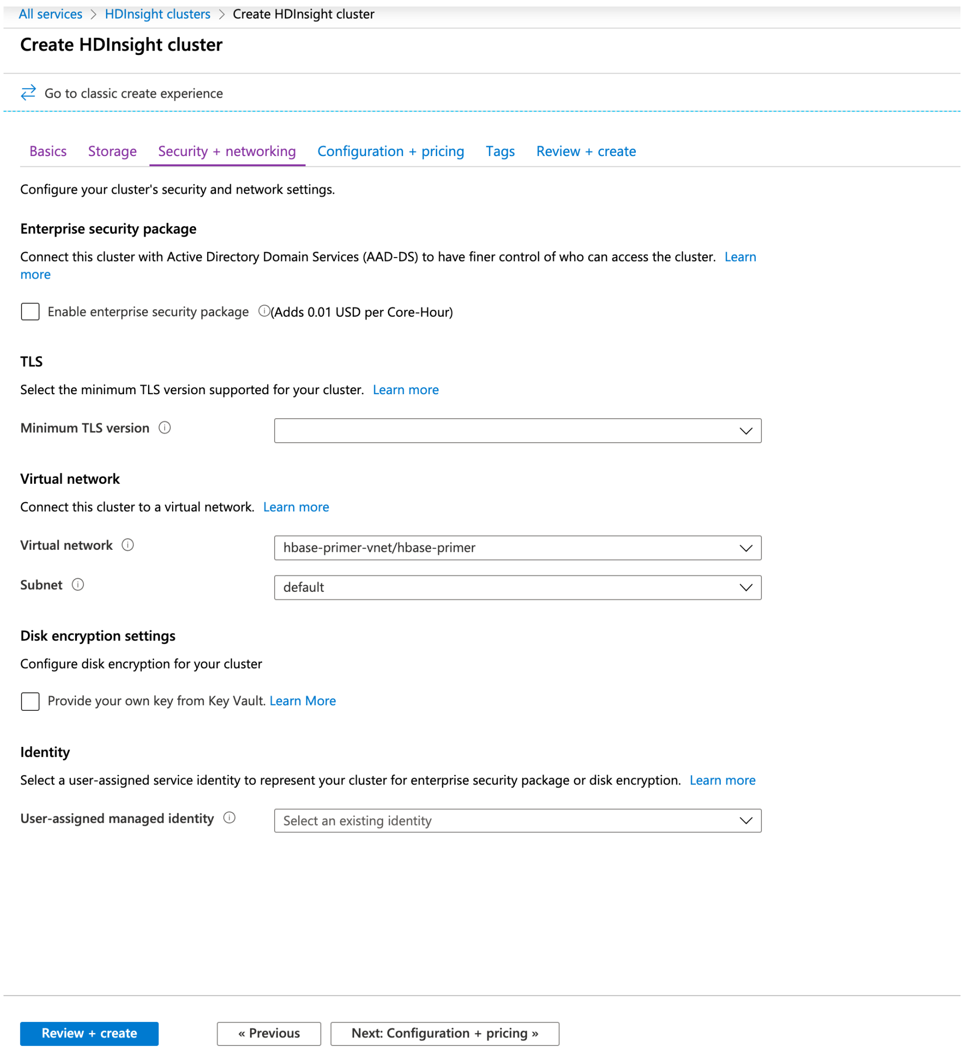
Enter new credentials that will be used to administer or access the cluster.

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

Figure 9: An example of the basic configuration for Azure HDInsight

4. In the "Storage" step:
    1. Select a storage account (or create a new one) and leave the others unchanged. A reminder that the Storage account name must be lowercase alphanumeric. A wrong Storage account name will cause the deployment of the cluster to fail.
5. In the "Security and Network" step:

1. Select the same Virtual Network and subnet as your VM. Please make sure that your HDInsight cluster is in the same virtual network as your VM. Failing to do so will prevent the VM from connecting to the HDInsight cluster during the following practice with the HBase Java API.



Configure your cluster's security and network settings.

Enterprise security package

Connect this cluster with Active Directory Domain Services (AAD-DS) to have finer control of who can access the cluster. Learn more

☐ Enable enterprise security package (Adds 0.01 USD per Core-Hour)

TLS

Select the minimum TLS version supported for your cluster. Learn more

Minimum TLS version:

Virtual network

Connect this cluster to a virtual network. Learn more

Virtual network: hbase-primer-vnet/hbase-primer

Subnet: default

Disk encryption settings

Configure disk encryption for your cluster

☐ Provide your own key from Key Vault. Learn More

Identity

Select a user-assigned service identity to represent your cluster for enterprise security package or disk encryption. Learn more

User-assigned managed identity: Select an existing identity

Figure 10: An example of the network configuration for Azure HDInsight

6. In the "Configuration + pricing" step, you may want to adjust the size and the numbers of nodes. Be aware of your budget, and Azure will show the hourly cost for confirmation before you launch the cluster. **Warning: your subscription will be disabled if you run out of the budget in your subscription. Please exercise caution and plan your budget.**
7. Do not forget to tag your resources with Key: `project` and Value as the tag value being used in the **currently ongoing project**.
8. Create the cluster in the "Review + create" step.

After the VM and the HBase cluster are ready, you can move on to the next section.

Loading data into HBase

HBase includes several methods of loading data into tables. The most straightforward method to load data is to either use **Bulk Loading** or to run your own MapReduce application. Bulk loading uses a built-in utility tool called `ImportTsv`, which will use less CPU and network resources than simply using the HBase API but is less flexible. Due to this pronounced limitation of `ImportTsv`, we provide you with a MapReduce application called `YetAnotherImportTsv` as the alternative.

`YetAnotherImportTsv` enables you to load data into HBase with flexible schema which is not supported by `ImportTsv`. You can also learn from the code in `YetAnotherImportTsv` to develop your own MapReduce task to load data into HBase.

You should follow the steps below to load data into an HBase cluster.

1. SSH into the master node of the HBase cluster.
    1. With Azure HDInsight, the command is `ssh {SSH_USERNAME}@${CLUSTER-ID}-ssh.azurehdinsight.net`, for example, `ssh sshuser@hbase-primer-test-01-ssh.azurehdinsight.net`. You can find the commands in the "SSH + Cluster login" tab of the cluster dashboard on the Azure portal.
2. Fetch the `million_songs_metadata.csv` dataset from the Azure Blob.

    ```
    wget https://clouddeveloper.blob.core.windows.net/datasets/hbase-basics/million_s
    ongs_metadata.csv
    ```

3. To help you understand the data in `million_songs_metadata.csv`, we offer you a sample schema if the data is loaded into MySQL instead:

Col#	Name	MySQL Type	Description	Example
1	`track_id`	`varchar(19)`	The **unique** track ID.	TRAAAAK128F9318786
2	`title`	`text`	The song title.	Scream
3	`song_id`	`text`	The song ID, note that a song can be associated with many tracks (with very slight audio differences).	SOBLFFE12AF72AA5BA
4	`release`	`text`	Album name from which the track was taken.	Adelitas Way
5	`artist_id`	`text`	The Echo Nest artist ID, this field is not empty.	ARJNIUY12298900C91
6	`artist_mbid`	`text`	The musicbrainz.org ID for this artist, please note this field may be empty!	6ae6a016-91d7-46cc-be7d-5e8e5d320c54
7	`artist_name`	`text`	Artist name	Adelitas Way
8	`duration`	`double`	Duration of the track in seconds.	213.9424
9	`artist_familiarity`	`double`	An estimate of how familiar an artist is to the world.	0.639902515496
10	`artist_hottnesss`	`double`	An estimation of how hot an artist is.	0.461318337541

Col#	Name	MySQL Type	Description	Example
11	`year`	`int(4)`	Song release year.	2009

4. Note that the HDFS path `input` and `/input` are two different paths, as the relative path `input` matches the absolute path `/user/hadoop/input` . Create a new directory in HDFS to store the CSV file you want to import, e.g., `/input` instead of using the default `/user/hadoop` directory to avoid potential permission issues. For example:

```
hadoop fs -mkdir /input
```

5. Put the dataset to HDFS.

```
hadoop fs -put million_songs_metadata.csv /input
```

6. To check if the file is successfully stored to HDFS, use the following command (or you may specify your own file path):

```
hadoop fs -ls /input/million_songs_metadata.csv
```

7. Open up the HBase shell with the command `hbase shell` and create a table called `songdata` with a column family named `data` (create command documentation (http://hbase.apache.org/book.html#shell_exercises)).

```
hbase shell
> create 'songdata', 'data'
```

8. You can use `describe 'songdata'` to get the table metadata. After this, you may exit the HBase shell using `exit` .

```
hbase shell
> describe 'songdata'
```

9. Below we specify the procedure to load data into HBase using `ImportTsv` and `YetAnotherImportTsv` separately.

Bulk loading

Bulk Loading consists of two main steps:

The first step of Bulk Loading is to ship the raw files formatted as HBase data files (**HFiles**) directly to the file system. This output format writes out data in HBase's internal storage format so that they can be later loaded very efficiently into the HBase cluster. One approach to preparing the data is to use the `importTsv` tool. `importTsv` is a custom MapReduce application that will by default load data in a TSV (Tab Separated Value) format into HBase. You can run the following command to create the required **HFiles**. (`HBASE_ROW_KEY` has to refer to the column of data in our file that we wish to designate as our rowkey.)

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE_ROW_KEY,da
ta:title,data:song_id,data:release,data:artist_id,data:artist_mbid,data:artist_name,da
ta:duration,data:artist_familiarity,data:artist_hotttnesss,data:year -Dimporttsv.separ
ator=, -Dimporttsv.bulk.output=/hfiles songdata /input/million_songs_metadata.csv
```

The second step of Bulk Loading is to import the prepared HFiles into the cluster. We will introduce the `completebulkload` tool here. There is a pitfall here that the data generated via MapReduce often has file permissions that are not compatible with the running HBase process. Since you log in the cluster as user `hadoop`, the data is also owned by `hadoop` and not accessible by `hbase`, the user used by the `completebulkload` tool. 1. You can check the current permission and ownership status by running this command:

```
    hadoop fs -ls /hfiles
```

1. To make it accessible to the user `hbase`, you should change the owner of the HFiles **recursively** to `hbase`, otherwise, the `completebulkload` command will get stuck.

   ```
   hadoop fs -chown -R hbase:hbase /hfiles
   ```

2. Now we can use this command to finish the importing:

   ```
   hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles /hfiles songdata
   ```

3. To verify you have successfully uploaded the data into the HBase table, you can open the HBase shell and scan the first 5 rows in the table:

   ```
   hbase shell
   > scan 'songdata', {'LIMIT' => 5}
   ```

4. Before trying other data loading methods, you can use the following command to clean the table records.

   ```
   > truncate 'songdata'
   ```

Non-bulk loading

`importTsv` also supports an alternative non-bulk loading way of loading data from HDFS into HBase using `Puts`.

```
    hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimport
tsv.columns=HBASE_ROW_KEY,data:title,data:song_id,data:release,data:artist_id,data:art
ist_mbid,data:artist_name,data:duration,data:artist_familiarity,data:artist_hottnesss,
data:year songdata /input/million_songs_metadata.csv
```

MapReduce loading

1. Download the source code of `YetAnotherImportTsv` with an environment where Maven is installed so that you can package the program as a JAR. You may install Maven on the HBase cluster on your own.

   ```
   wget https://clouddeveloper.blob.core.windows.net/f21-cloud-developer/hbase/YetAn
   otherImportTsv.tgz

   tar -xvzf YetAnotherImportTsv.tgz
   ```

2. Read the source code and update the configuration. The `hbase.zookeeper.quorum` property is a comma-separated list of hosts on which ZooKeeper servers are running.
   1. For an EMR cluster, the ZooKeeper address is the private IP of the master node.
   2. For an HDInsight cluster, go to the HDInsight dashboard and click the "ZooKeeper" tab in the sidebar. You can find that there are 3 ZooKeeper nodes shown in the HDInsight dashboard, click each of them to get the private IP of ZooKeeper servers. As there are 3 ZooKeeper nodes on HDInsight, the value of `hbase.zookeeper.quorum` should be `private_ip_1,private_ip_2,private_ip_3`.
   3. **Warning**: Do not forget to update the `inputPath` and `tableName` in the source code.

Configuration for AWS EMR:

```
Configuration conf = HBaseConfiguration.create();
// TODO: update the Zookeeper address
String zkAddr = "";
conf.set("hbase.master", zkAddr + ":14000");
conf.set("hbase.zookeeper.quorum", zkAddr);
conf.set("hbase.zookeeper.property.clientport", "2181");
```

Configuration for Azure HDInsight:

```
Configuration conf = HBaseConfiguration.create();
// TODO: update the Zookeeper address
String zkAddr = "";
conf.set("hbase.zookeeper.quorum", zkAddr);
conf.set("hbase.zookeeper.property.clientport", "2181");
conf.set("hbase.cluster.distributed", "true");
// Linux-based HDInsight clusters use /hbase-unsecure as the znode parent
conf.set("zookeeper.znode.parent","/hbase-unsecure");
```

1. Package the jar.

   ```
   mvn clean package
   ```

2. Run the MapReduce application to load the data.

   ```
   yarn jar target/import_tsv.jar edu.cmu.cc.utils.YetAnotherImportTsv
   ```

HBase Compaction

HBase is a distributed wide column database, designed to perform random read/writes on billions of rows multiply millions of columns. Optimal read performance comes from having one file per column family. It is not always possible to have one file per column family during heavy writes. That is the reason why HBase tries to combine all HFiles into a large single HFile to reduce the maximum number of disk seeks needed for read. This process is known as compaction. Compactions choose some files from a single store in a region and combine them. This process involves reading KeyValues in the input files and writing out any KeyValues that are not deleted, are inside of the time to live (TTL), and don't violate the number of versions. The newly created combined file then replaces the input files in the region. Besides, as mentioned in the early HBase architecture section, compaction can help HBase achieve locality. There are two types of HBase compaction:

- **Minor Compaction**: Combines a configurable number of small files and merges them into a bigger file. Minor compaction is less resource intensive than Major compaction and takes less time, but it happens more frequently.
- **Major Compaction**: Major compaction is a process of combining the StoreFiles of regions into a single StoreFile. It also deletes removed and expired versions. By default, major compaction runs every 24 hours and merges all StoreFiles into single StoreFile. After compaction, if the new larger StoreFile is greater than a certain threshold (defined by property), the region will split into new regions. Then the parent HRegion will go offline and the two sub-HRegions are assigned to the corresponding HRegionServer by HMaster.

If you let HBase accumulate many HFiles without compacting them, you'll achieve better write performance (the data is rewritten less frequently) but get worse read performance (reading particular rows requires many disk reads and can reduce overall performance). If on the other hand you instruct HBase to compact many HFiles into a single HFile you'll have better read performance since you only need to read one file and there is less disk read.

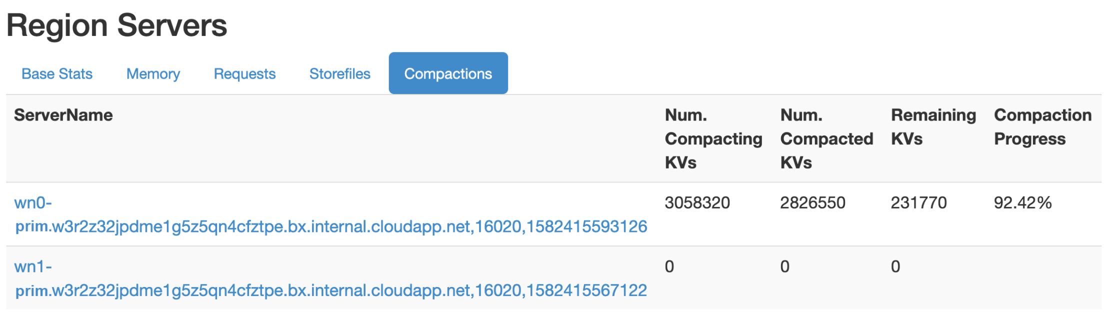
You can run the following operations in the HBase shell to see how major compaction can improve the query performance. Run the first query below, record the execution shown in the result.

```
> scan 'songdata', {COLUMNS => ['data:artist_name', 'data:title'], FILTER => "SingleCo
lumnValueFilter('data', 'artist_name', = , 'regexstring:^The Beatles.*')"}
```

Apply major compaction to the table by using the command `major_compact`.

```
> major_compact 'songdata'
```

Please note that the major compaction does not finish immediately when the HBase shell command returns some results. You can monitor the compaction progress in HBase UI (refer to this link (https://docs.microsoft.com/en-us/azure/hdinsight/hbase/apache-hbase-tutorial-get-started-linux) to learn how to access the HBase Master UI). Besides monitoring the compaction progress metrics, you should also monitor how the locality changes for each region during the compaction.



ServerName	Num. Compacting KVs	Num. Compacted KVs	Remaining KVs	Compaction Progress
wn0-prim.w3r2z32jpdme1g5z5qn4cfztpe.bx.internal.cloudapp.net,16020,1582415593126	3058320	2826550	231770	92.42%
wn1-prim.w3r2z32jpdme1g5z5qn4cfztpe.bx.internal.cloudapp.net,16020,1582415567122	0	0	0	

Figure 11: HBase data compaction monitor

After finishing the major compaction, rerun the previous query, and you can see that the major compaction can greatly reduce the lookup time.

```
> scan 'songdata', {COLUMNS => ['data:artist_name', 'data:title'], FILTER => "SingleCo
lumnValueFilter('data', 'artist_name', = , 'regexstring:^The Beatles.*')"}
```

Compression and Data Block Encoding in HBase

HBase has many options to encode or compress the data on disk. In this section, we will explore compression and block encoding.

Data compression

Compression means full compression of HBase blocks by reducing the size of large and opaque byte arrays in cells. Using compression with HBase reduces the number of bytes transmitted over the network and stored on disk. These benefits often outweigh the performance cost of compressing the data on every write and uncompressing it on every read.

HBase provides 4 kinds of compressors: **Snappy**, **LZO**, **LZ4**, **GZ**. To use compression, you can specify it while creating or altering tables in HBase shell. In the following example, `songdata` is the table name, `data` is the column family name.

```
hbase shell
> create 'songdata', { NAME => 'data', COMPRESSION => 'GZ' }

# change compression
> alter 'songdata',  { NAME => 'data', COMPRESSION => 'SNAPPY' }
> major_compact 'songdata'
```

Data block encoding

Block encoding refers to the ability to encode KeyValues or Cells on-the-fly as blocks are written or read, exploiting duplicate information between consecutive Cells. HBase stores each cell individually, with its key and value, when a row has many cells, much space would be consumed by writing the same key many times. Therefore, encoding can save much space especially for large rows.

HBase provides four data block encoding types: **PREFIX**, **DIFF**, **FAST_DIFF**, **PREFIX_TREE**. To enable data block encoding, you can set the `DATA_BLOCK_ENCODING` property for the table. In the following example, `songdata` is the table name, `data` is the column family name.

```
hbase shell
> disable 'songdata'
> alter 'songdata', { NAME => 'data', DATA_BLOCK_ENCODING => 'FAST_DIFF' }
> enable 'songdata'
> major_compact 'songdata'
```

Information

Notes

1. Compression and data block encoding can be used together on the same ColumnFamily.
2. If you change compression or encoding for a ColumnFamily, the changes only take effect during compaction.
3. Refer to HBase Reference Guide (http://hbase.apache.org/book.html#compression) to learn more details about the difference between these compression algorithms and data block encoding types. Besides, this guide answers which compressor or data block encoder to use depends on the characteristics of the data.

HBase Data Migration

There are multiple methods that can be used to migrate an HBase table between different clusters. In this primer, we want to introduce 2 approaches: **snapshot** and **export/import**.

Export/Import

Export is a utility that will dump the contents of the table to HDFS in a sequence file via MapReduce. The following example shows how to migrate the table `songdata` into another cluster. 1. Export the table content to HDFS. Here, `songdata` is the name of the table, `songdata_dir` is the HDFS directory where we want to export our data.

```
    hbase org.apache.hadoop.hbase.mapreduce.Export songdata /songdata_dir
```

1. Download the exported table data from HDFS to local. Here, `/songdata_dir` is the path in HDFS, and `./` is the path in the local server.

   ```
   hadoop fs -get /songdata_dir ./
   ```

2. Transmit the downloaded data to another cluster by using `scp` or other methods.

```
scp -r ./songdata_dir sshuser@new-hbase-cluster-ssh.azurehdinsight.net:~/
```

3. Connect to the target cluster, upload the transmitted data to HDFS. Here, `./songdata` is the path in the local server while `/` is the root path of HDFS.

```
hadoop fs -put ./songdata_dir /
```

4. Before importing, you need to guarantee that the target table exists in HBase, since importing is an append process rather than overwrite process.

```
hbase shell
> create 'songdata', 'data'
> exit
```

5. Import the table content into the HBase. In the following command, `songdata` is the name of the new table, and `/songdata_dir` is the HDFS directory that stores the table content.

```
hbase org.apache.hadoop.hbase.mapreduce.Import songdata /songdata_dir
```

Snapshot

A snapshot is a set of metadata information that allows an admin to get back to a previous state of the table. HBase snapshot support enables you to take a snapshot of a table without much impact on RegionServers, because snapshot, clone, and restore operations do not involve data copying. In addition, exporting a snapshot to another cluster has no impact on RegionServers.

The following commands show how to use snapshot operations in HBase shell:

```
hbase shell

# take a snapshot on a specified table.
> snapshot 'tableName', 'snapshotName'

# list all of the snapshots
> list_snapshots

# delete a specified snapshot
> delete_snapshot 'snapshotName'

# clone a table from snapshot.
> clone_snapshot 'snapshotName', 'newTableName'

# Replace the current table with a specified snapshot content.
# The restore operation requires the table to be disabled.
> disable 'tableName'
> restore_snapshot 'snapshotName'
```

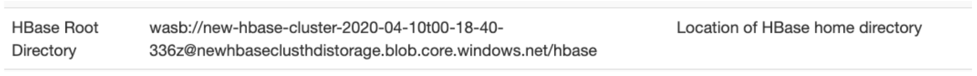
To export a snapshot to another cluster, you can use the `ExportSnapshot` tool which copies all the data related to a snapshot (HFiles, logs, snapshot metadata) to another cluster. This tool executes a Map-Reduce job to copy files between the two clusters. Since it works at file-system level the hbase cluster does not have to be online. The following command shows copying a snapshot called "snapshotName" from an HBase cluster srv1 to another HBase cluster srv2

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot \
    -snapshot snapshotName -copy-from hdfs://srv1:8082/hbase \
    -copy-to hdfs://srv2:8082/hbase
```

Information

Notes

1. To export the snapshot, you need to specify the HDFS location, which can be found from the HBase Master UI.



HBase Root Directory	wasb://new-hbase-cluster-2020-04-10t00-18-40-336z@newhbaseclusthdistorage.blob.core.windows.net/hbase	Location of HBase home directory

Figure 12: HDFS root directory

2. You need to guarantee that the snapshot stored in the source HBase cluster could be accessed by the target cluster. For example, if you are using Azure, you need to modify the access level of the container that stores the snapshot.
3. You should run the `ExportSnapshot` from the target HBase cluster, since in most of the case, HBase cluster doesn't expose the write permission to other clusters.

Danger

Troubleshooting

There is a known bug in HBase that `ExportSnapshot` will fail if copying files to root directory takes longer than cleaner TTL (refer to this JIRA ticket (https://issues.apache.org/jira/browse/HBASE-23202) for more details). One solution of this bug is to disable HFile cleaner before snapshot import and enable it after finishing data migration. The HBase command `cleaner_chore_switch` can be used to enable and disable the cleaner.

Bloom Filter

A Bloom filter is a data structure which is designed to predict whether a given element is a member of a set of data. A positive result from a Bloom filter is not always accurate, but a negative result is guaranteed to be accurate. For more information about Bloom filters in general, refer to Bloom filter - WikiPedia (https://en.wikipedia.org/wiki/Bloom_filter).

In terms of HBase, Bloom filters provide a lightweight in-memory structure to reduce the number of disk reads for a given `GET` operation to only the StoreFiles likely to contain the desired Row. The Bloom filters themselves are stored in the metadata of each HFile and never need to be updated. When a HFile is opened, typically when a region is deployed to a RegionServer, the Bloom filter is loaded into memory.

HBase provides two types of Bloom filters: `ROW` and `ROWCOL` . See When To Use Bloom Filters (http://hbase.apache.org/book.html#schema.bloom) for more information on `ROW` versus `ROWCOL` . The following example shows how to enable Bloom filter.

```
hbase shell
> create 'mytable',{NAME => 'colfam1', BLOOMFILTER => 'ROWCOL'}
> describe 'mytable'

# alter the Bloom filter into ROW
> alter 'mytable', { NAME => 'colfam1', BLOOMFILTER => 'ROW' }
> describe 'mytable'
```

HBase Query

HBase Query

HBase offers a shell where you can query the data along with the filtering tools. The data is stored in columns, specified by the column qualifier. Columns are grouped into column families, which in our case is `data` . HBase is very good at matching prefixes because rows are sorted lexicographically by rowkey, in this section we will explore it through "scan" queries.

The following semantics show the basic structure of how to make a query in the HBase shell:

```
> scan 'table_name', {COLUMNS => ['column1', 'column2', …], FILTER => "(FILTER1) … (FI
LTER2)"}
```

Suppose we want to find all the tracks whose `artist_name` begin with "The Beatles". Note that this is a prefix match instead of a substring match. The query in the HBase shell would look like:

```
> scan 'songdata', {COLUMNS => 'data:artist_name', FILTER => "SingleColumnValueFilter
('data', 'artist_name', = , 'regexstring:^The Beatles.*')"}
```

Note that the columns are specified as `(column_family_name):(column_qualifier_name)` .

`data:artist_name` will be the only column returned by the query above, because we only specified `data:artist_name` as the arguments of `COLUMNS` in the query. If we want to retrieve more columns of each row, we need to add the columns as the arguments of `COLUMNS` :

```
> scan 'songdata', {COLUMNS => ['data:artist_name', 'data:title'], FILTER => "SingleCo
lumnValueFilter('data', 'artist_name', = , 'regexstring:^The Beatles.*')"}
```

Similar to how we specify more columns as the arguments of `COLUMNS` , we can also specify multiple `FILTER` arguments. Filters can be combined using logical operators like `AND` , `OR` , `WHILE` , etc. For example, if we wanted to modify the previous query which gets the songs whose `artist_name` begin with "The Beatles", so that it only retain the rows whose title begins with a `W` or a character lexicographically greater than `W` , we should use the following two filters joined by an `AND` operator:

```
> scan 'songdata', {COLUMNS => ['data:artist_name', 'data:title'], FILTER => "SingleCo
lumnValueFilter('data', 'artist_name', = , 'regexstring:^The Beatles.*') AND SingleCol
umnValueFilter('data', 'title', >= , 'binaryprefix:W')"}
```

Please note that whenever you use a FILTER on a column, **the column must be added to the COLUMNS list**, or the filter will not apply.

You may also perform a range prefix scan with `STARTROW` and `ENDROW` .

```
> scan 'table_name', {STARTROW => 'start_row', ENDROW => 'end_row'}
```

Note that the scan ranges will be `STARTROW <= key < ENDROW`.

Information

Notes

For more information about the HBase filter, please refer to this post (http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/admin_hbase_filtering.html). For more information about how HBase Scan works and Scan performance tuning, please refer to this article (https://www.dotkam.com/2015/10/08/hbase-scan-let-me-cache-it-for-you/).

HBase Java API

HBase Java API

The HBase Java API allows you to create, list, alter or delete tables, as well as insert or query data from tables. In this section, you will explore how to work with HBase Java API.

Establish Connections

You need to establish a connection to the HBase cluster before you make queries. Here is a simple example of how to connect to HBase using Java.

Configuration for AWS EMR:

```
Configuration conf = HBaseConfiguration.create();
// TODO: update the Zookeeper address
String zkAddr = "";
conf.set("hbase.master", zkAddr + ":14000");
conf.set("hbase.zookeeper.quorum", zkAddr);
conf.set("hbase.zookeeper.property.clientport", "2181");
```

Configuration for Azure HDInsight:

```
Configuration conf = HBaseConfiguration.create();
// TODO: update the Zookeeper address
String zkAddr = "";
conf.set("hbase.zookeeper.quorum", zkAddr);
conf.set("hbase.zookeeper.property.clientport", "2181");
conf.set("hbase.cluster.distributed", "true");
// Linux-based HDInsight clusters use /hbase-unsecure as the znode parent
conf.set("zookeeper.znode.parent","/hbase-unsecure");
```

You can then choose the HBase table.

```
Connection conn = ConnectionFactory.createConnection(conf);
TableName tableName = TableName.valueOf("songdata");
Table songsTable = conn.getTable(tableName);
```

Accessing HBase Data

There are two common ways that a client interacts with HBase tables: `SCAN` and `GET`. `SCAN` is usually used to retrieve multiple rows with filters. It needs to scan the whole table, so `SCAN` is an `O(N)` operation where `N` is the total number of lines in the table. `GET` is used to search and retrieve one single row by the rowkey. It uses Bloom filters to skip StoreFiles that do not contain a certain row key, thus it is much faster than the `SCAN` operation. Since you can only specify a rowkey for `GET`, the HBase schema should be carefully designed if you want to optimize the throughput using `GET`. You may also specify the `STARTROW` and `ENDROW` of the `SCAN` operation to largely improve the performance of `SCAN`.

In this section, you will be using `SCAN` to find out which row(s) match specific filtering rules.

Below is a simple example to print all the `artist_name` entries that start with "The Beatles".

For more information on GET (https://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Get.html) and SCAN (https://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Scan.html), please refer to the HBase Java API documentation (https://hbase.apache.org/apidocs/index.html) and the hbase.apache.org book (http://hbase.apache.org/book.html#_java_2).

```
// Create a new Scan object.
// By calling the default constructor, the entire table will be scanned.
Scan scan = new Scan();

// The binary representation of the column family name.
byte[] bColFamily = Bytes.toBytes("data")

// The binary representation of the column name.
byte[] bCol = Bytes.toBytes("artist_name");

// This is used for regular expression matching.
// You should use different comparators based on specific requirements.
RegexStringComparator comp = new RegexStringComparator("^The Beatles.*");

// The filtering rules of the Scan object.
Filter filter = new SingleColumnValueFilter(bColFamily, bCol, CompareFilter.CompareOp.
EQUAL, comp);

// Associate the filtering rules to the Scan object.
scan.setFilter(filter);

// Get the scan result.
ResultScanner rs = songsTable.getScanner(scan);

// Each call of rs.next() will return one row.
for (Result r = rs.next(); r != null; r = rs.next()) {
    // r represents one row in the table.
    // r.getValue returns the specific cell (determined by column family
    // and column name).
    System.out.println(Bytes.toString(r.getValue(bColFamily, bCol)));
}

// Cleanup
rs.close();
```

To better complete the tasks in your project, you should also read the docs on HBase Client Request Filters (http://hbase.apache.org/book.html#client.filter) and the Filter class (http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/filter/Filter.html).

Security Best Practice of Hadoop clusters

Security Best Practice of Hadoop clusters on AWS EMR

The EMR team confirmed an issue where users are opening the firewall (SecurityGroup) of their EMR clusters to the internet, and the clusters get exploited and victimized to be implicated in DDOS attacks.

Hadoop clusters in their default configuration provide unauthenticated remote code execution via the Hadoop/YARN APIs, which expose security vulnerabilities such as Denial of Service attacks, and other hacks.

No EMR release is vulnerable to remote code execution in its *default* configuration with the *default* EMR-provided SecurityGroup settings. However, the vulnerabilities will be exposed by opening EMR service ports to the public.

To address the security best practice of Hadoop clusters, **we require you to review the SecurityGroups** for any open Hadoop service ports and close them ASAP.

You should **NEVER** open all the ports to the public (0.0.0.0) when using instances on a public cloud.

AWS suggests using SSH tunneling to access the Web UI on the port 8088 which is known to have security vulnerabilities if made public. We outline the steps that you can follow to access the Web UI without opening port 8088 to the public (0.0.0.0).

Visit the YARN UI without opening unsafe ports to the public

1. Set up an SSH tunnel using dynamic port forwarding. You should replace port 8123 with an unused port on your local computer.

   ```
   ssh -i ~/mykeypair.pem -N -D 8123 hadoop@ec2-###-##-##-###.compute-1.amazonaws.com
   ```

   The terminal will remain open and will not return a response after you run this command.

2. You must use a SOCKS proxy management tool to control the proxy settings of your browser when you use an SSH tunnel with proxy forwarding. AWS suggests using Foxy Proxy which is available on Firefox, Chrome, and IE. Please follow this link for Google Chrome https://chrome.google.com/webstore/search/foxy%20proxy (https://chrome.google.com/webstore/search/foxy%20proxy). You can find the relevant links for the browser you are using.

3. Now, create a file `foxyproxy-settings.xml` using a text editor with the following content. Make sure that you are using the port on which you opened the tunnel instead of 8123.

```
<?xml version="1.0" encoding="UTF-8"?>
<foxyproxy>
  <proxies>
    <proxy name="emr-socks-proxy" id="2322596116" notes="" fromSubscription="fals
e" enabled="true" mode="manual" selectedTabIndex="2" lastresort="false" animatedI
cons="true" includeInCycle="true" color="#0055E5" proxyDNS="true" noInternalIPs
="false" autoconfMode="pac" clearCacheBeforeUse="false" disableCache="false" clea
rCookiesBeforeUse="false" rejectCookies="false">
      <matches>
        <match enabled="true" name="*ec2*.amazonaws.com*" pattern="*ec2*.amazonaw
s.com*" isRegEx="false" isBlackList="false" isMultiLine="false" caseSensitive="fa
lse" fromSubscription="false" />
        <match enabled="true" name="*ec2*.compute*" pattern="*ec2*.compute*" isRe
gEx="false" isBlackList="false" isMultiLine="false" caseSensitive="false" fromSub
scription="false" />
        <match enabled="true" name="10.*" pattern="http://10.*" isRegEx="false" i
sBlackList="false" isMultiLine="false" caseSensitive="false" fromSubscription="fa
lse" />
        <match enabled="true" name="*10*.amazonaws.com*" pattern="*10*.amazonaws.
com*" isRegEx="false" isBlackList="false" isMultiLine="false" caseSensitive="fals
e" fromSubscription="false" />
        <match enabled="true" name="*10*.compute*" pattern="*10*.compute*" isRegE
x="false" isBlackList="false" isMultiLine="false" caseSensitive="false" fromSubsc
ription="false" />
        <match enabled="true" name="*.compute.internal*" pattern="*.compute.inter
nal*" isRegEx="false" isBlackList="false" isMultiLine="false" caseSensitive="fals
e" fromSubscription="false"/>
        <match enabled="true" name="*.ec2.internal* " pattern="*.ec2.internal*" i
sRegEx="false" isBlackList="false" isMultiLine="false" caseSensitive="false" from
Subscription="false"/>
      </matches>
      <manualconf host="localhost" port="8123" socksversion="5" isSocks="true" us
ername="" password="" domain="" />
    </proxy>
  </proxies>
</foxyproxy>
```

4. Choose Options in the Foxy Proxy addon.
5. Choose **Import/Export** on the Foxy Proxy page.
6. On the **Import/Export** page, choose **Choose File**, browse to the location of the foxyproxy-settings.xml file you created, select the file, and choose **Open**.
7. Choose Replace when prompted to overwrite the existing settings.
8. For **Proxy mode**, choose Use **proxies based on their predefined patterns and priorities**.
9. To open the web interfaces, in your browser's address bar, type master-public-dns followed by the port number. (Eg. **http://ec2-xx-xxx-xxx-xxx.compute-1.amazonaws.com:8088**)

Access an HBase Master Node from a VM outside the cluster

1. **You have to select the same subnet as the HBase Cluster when you create your VM.** Since you need to use the **Private IP** to access your HBase Master Node, you can only use a VM which is in the same subnet as the HBase cluster.
2. *(Optional)* Note that if you are using the default EMR security group, you can skip this step. If you are using a custom security group for your HBase Master Node, add an inbound rule in your custom security group to open all the ports to itself.

   2.1. Click the “Edit” button in the Inbound tab.

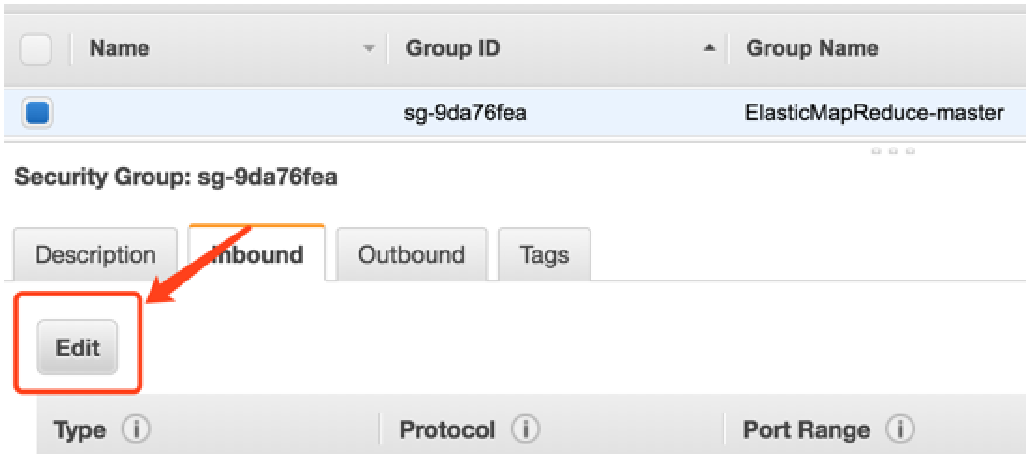
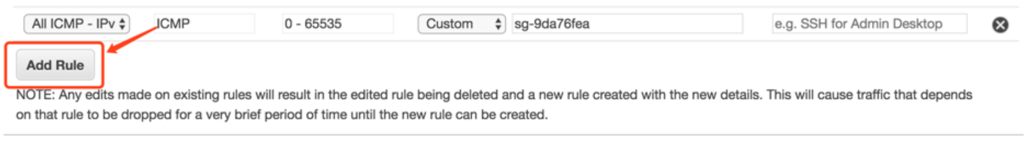


Figure 13: Edit the security group of the HBase Master Node

2.2. Click “Add Rule” to add a new inbound rule.



NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffic that depends on that rule to be dropped for a very brief period of time until the new rule can be created.

Figure 14: Add an inbound rule to the security group of the HBase Master Node

2.3. Select All TCP traffic, input `0 - 65535` to the port range and put its own Group ID (shown in the security group list) to the source field. It will whitelist any VMs which are assigned to this security group.

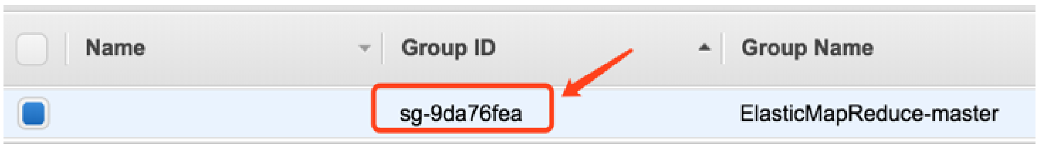


Figure 15: Group ID in the security group list

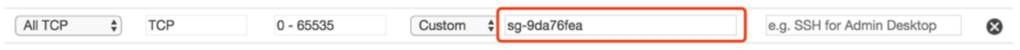


Figure 16: Put the group ID in the source filed

3. Add the security group of the Master Node to the VM you want to whitelist. Visit the EC2 web console, select the VM and click Actions - Change Security Groups. Then, choose the checkbox of the security group you used on your HBase Master Node.

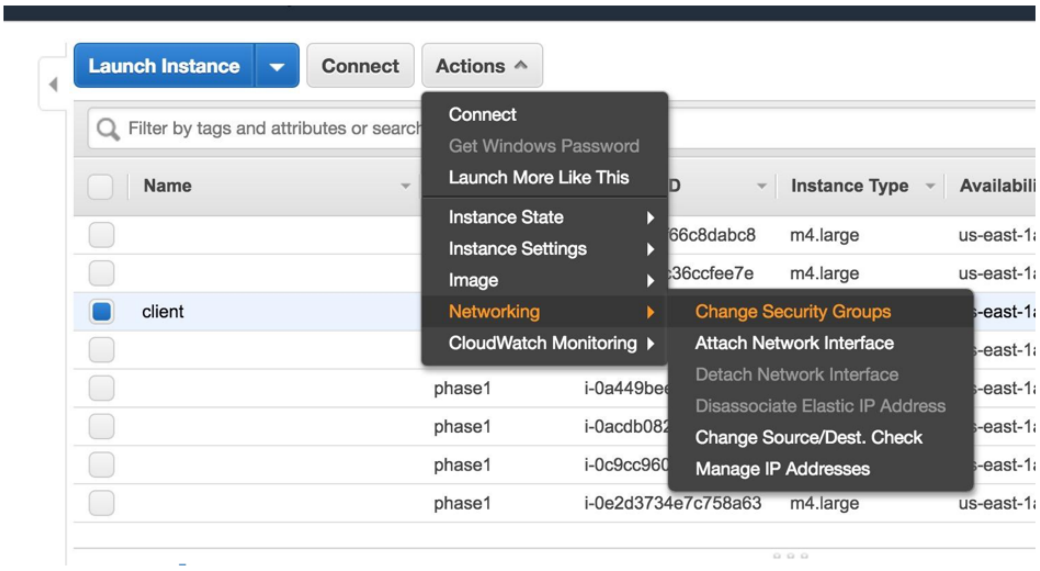
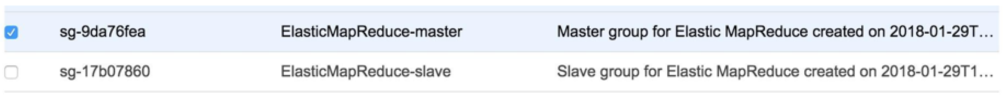


Figure 17: Change the security group of a VM



☑	sg-9da76fea	ElasticMapReduce-master	Master group for Elastic MapReduce created on 2018-01-29T…
☐	sg-17b07860	ElasticMapReduce-slave	Slave group for Elastic MapReduce created on 2018-01-29T1…

Figure 18: Add the security group of the Master Node to your outside VM

4. Make sure to use the **Private IP** of the HBase Master Node to access the HBase Cluster when you are programming on the VM.

Danger

Note that a 100% penalty will incur if you keep all the ports of the EMR cluster open to the **public** in the team project effectively this week, except **22, 80, 25, 443, or 465**.

The AWS abuse team may shut down your instances without notifications if your cluster gets victimized to be implicated in DDOS attacks in the future.

The default security group is allowed. You are allowed to open any ports **to the internal network**, e.g., you may whitelist the private IPs of your own remote/local machines to access all the ports as described above.

You can refer to the following for more detailed information

- https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-web-interfaces.html (https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-web-interfaces.html)
- https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-ssh-tunnel.html (https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-ssh-tunnel.html)
- https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-connect-master-node-proxy.html (https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-connect-master-node-proxy.html)

Information

How about Azure HDInsight and GCP Dataproc?

If you are using GCP Dataproc clusters, you also need to exercise caution when you need to access the Hadoop UI and make sure the discussed ports are not open in the firewall rules. If your ports are open, your cluster will be compromised and likely used in a DDOS attack.

Azure HDInsight requires a password to access the cluster from the browser, which protects you from opening these ports by mistake. You do not need to open any ports to the public to access the pre-installed Ambari dashboard of an HDInsight cluster.