Big Data Analytics

Isara Anantavrasilp

Lecture 6: Installing Hadoop Distribution

Hadoop Distribution

- Hadoop distribution: Cloudera QuickStart
- Platform: Virtual Box
- System Requirements
  - 64-bit host OS and a virtualization that support 64-bit guest OS
  - RAM for VM: 4 GB
  - HDD: 20 GB

Installing Cloudera QuickStart

- Download size: ~5.5 GB
- Download links
  - https://www.virtualbox.org/wiki/Downloads
    Select package corresponding to your host system
  - https://downloads.cloudera.com/demo_vm/virtualbox/cloudera-quickstart-vm-5.13.0-0-virtualbox.zip

VirtualBox Download

Download VirtualBox

Here you will find links to VirtualBox binaries and its source code.

VirtualBox binaries

By downloading, you agree to the terms and conditions of the respective license.

If you're looking for the latest VirtualBox 5.1 packages, see VirtualBox 5.1 builds. Consider upgrading.

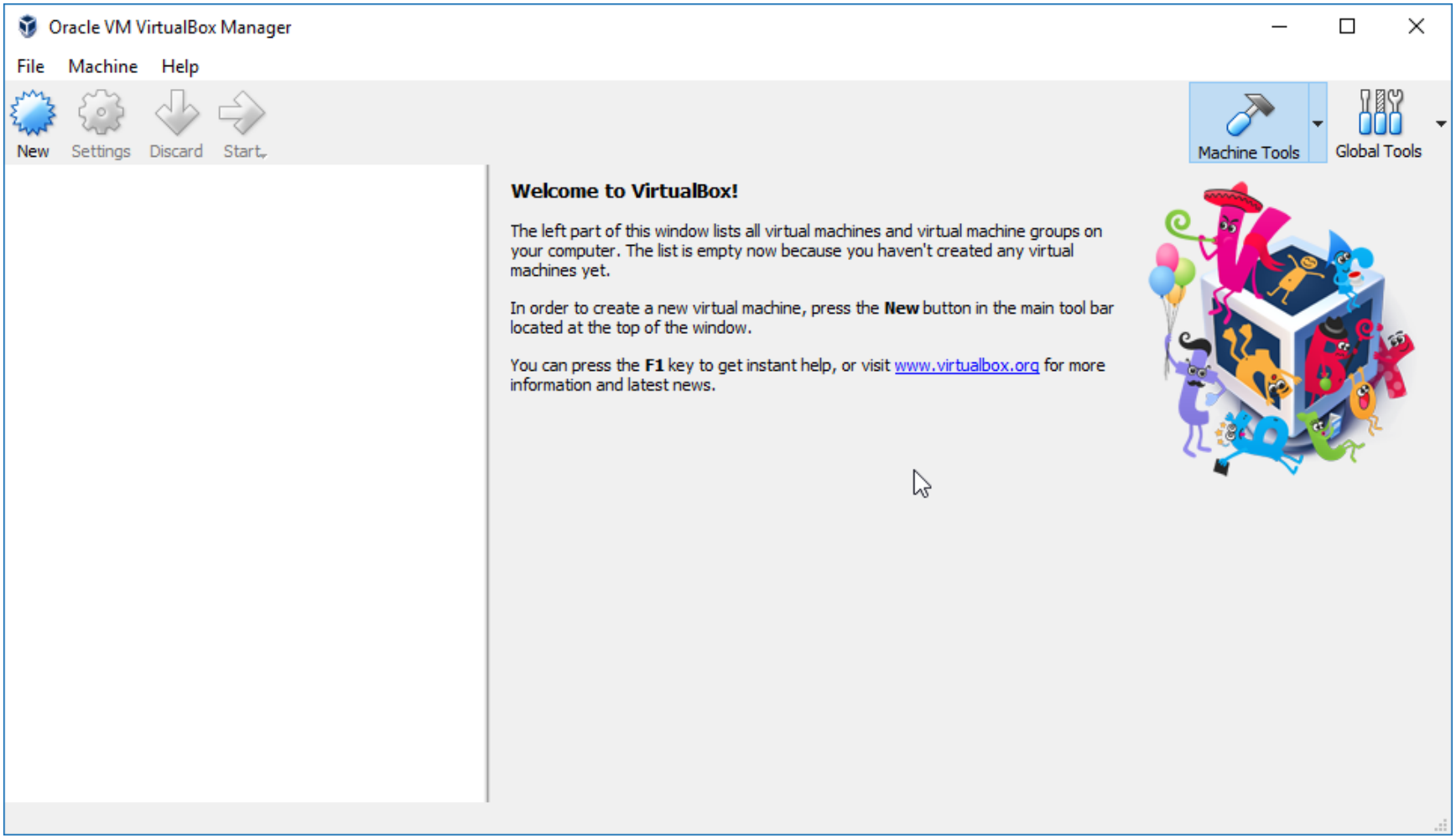
VirtualBox 5.2.18 platform packages

- Windows hosts
- OS X hosts
- Linux distributions
- Solaris hosts

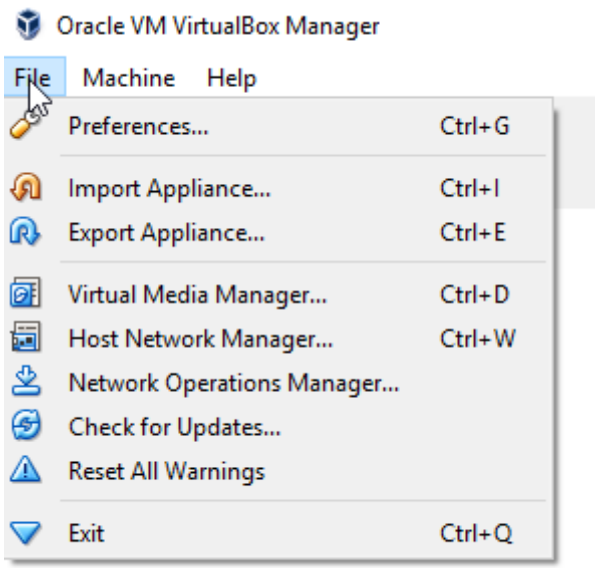
Installing Cloudera QuickStart

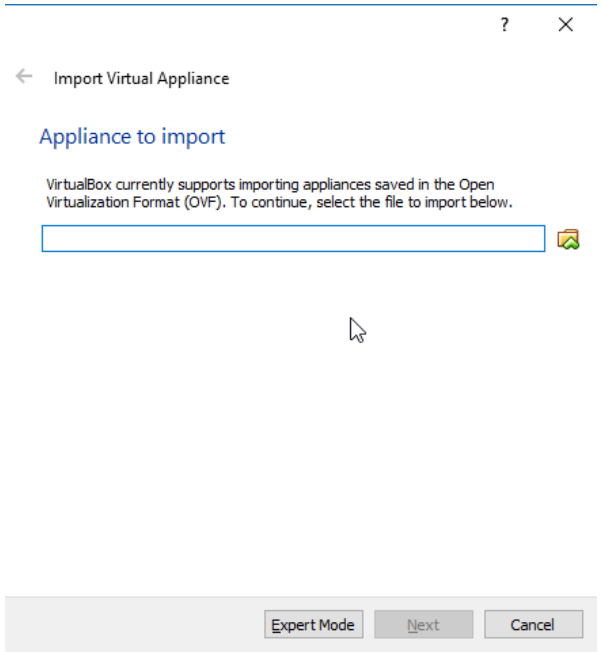
- Install VirtualBox
- Unzip Cloudera VM
- Start VirtualBox
- Import Appliance (Virtual Machine)
- Launch Cloudera VM

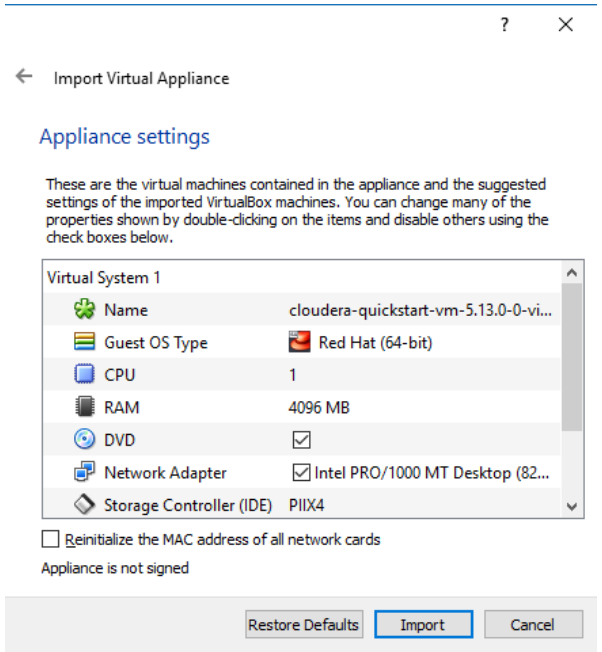
Start Virtual Box

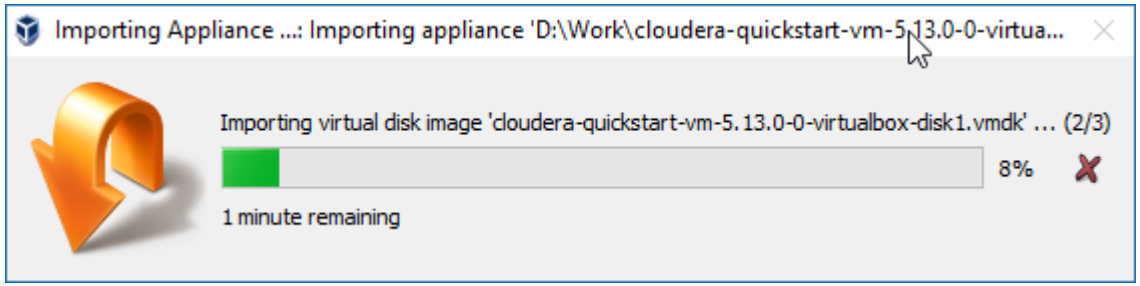


Import Appliance

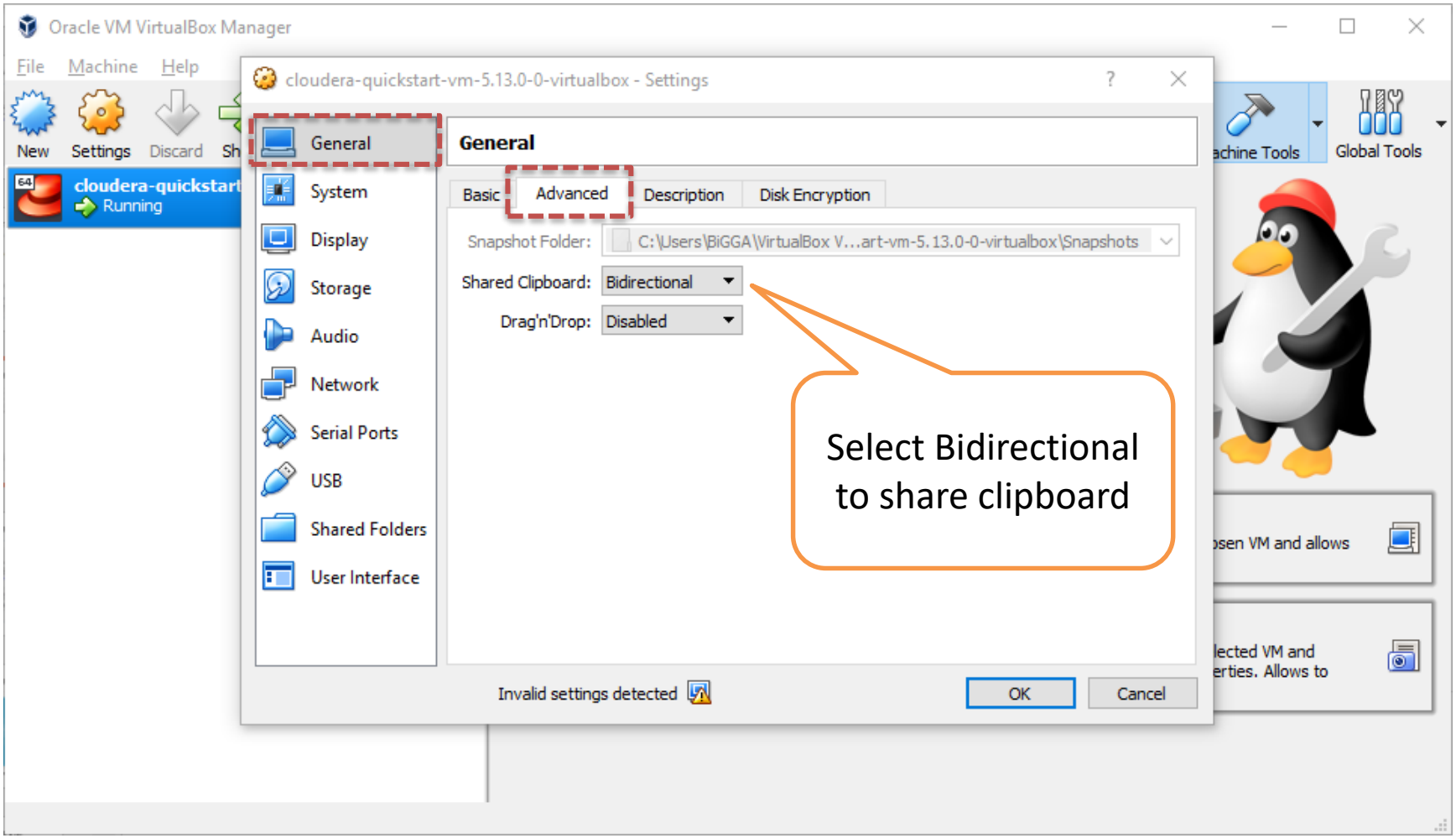




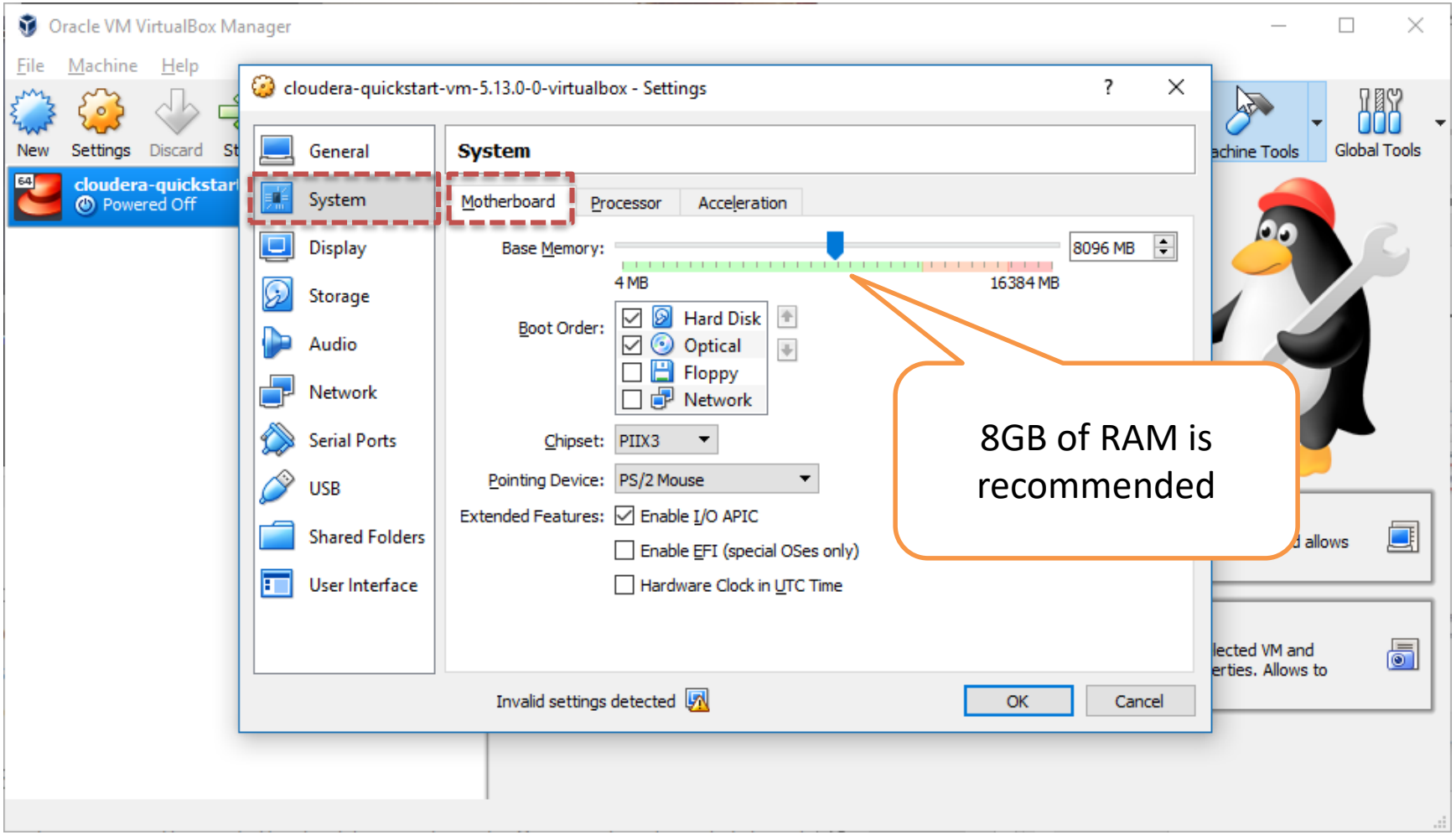




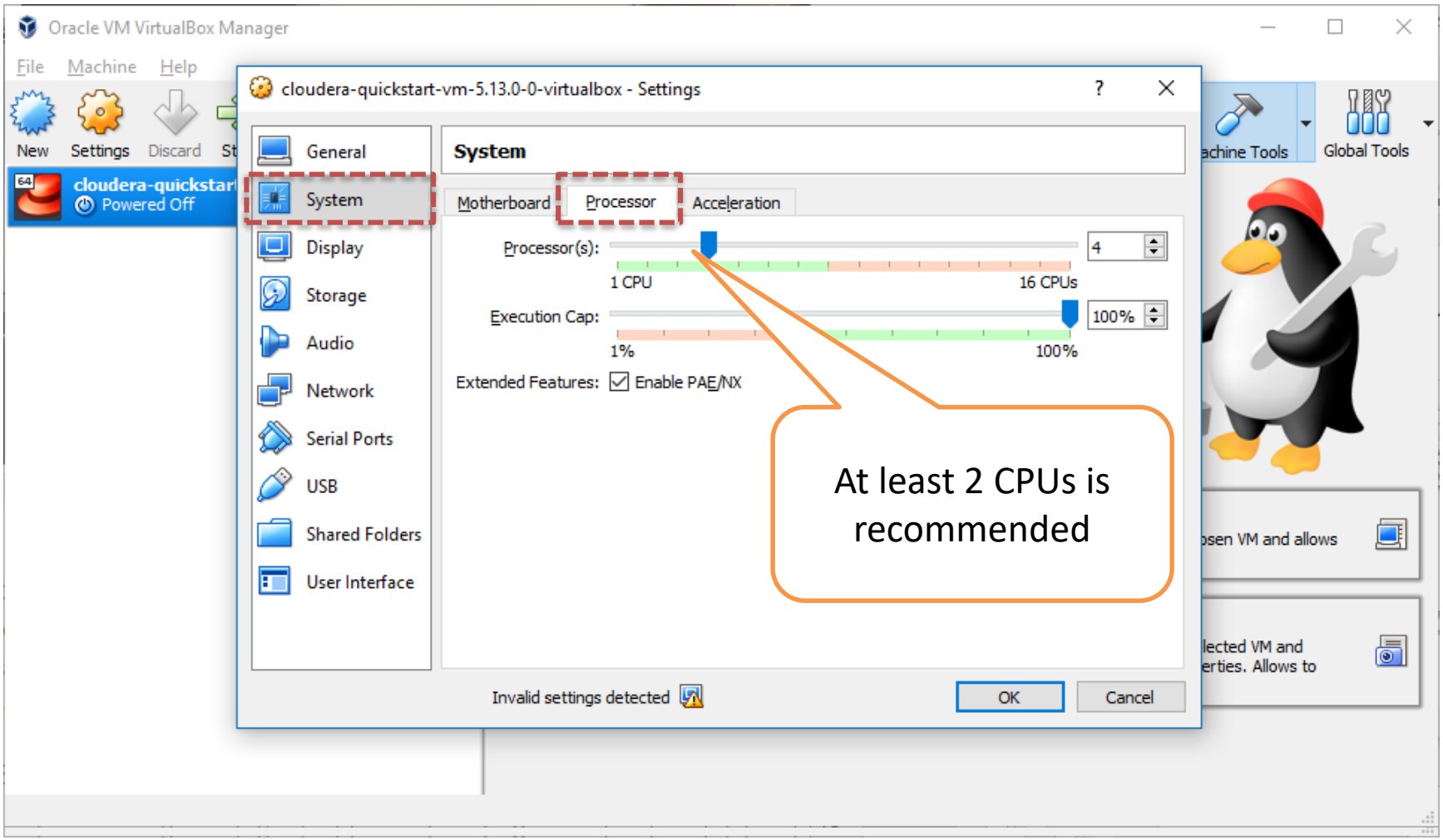
Setting Up the VM



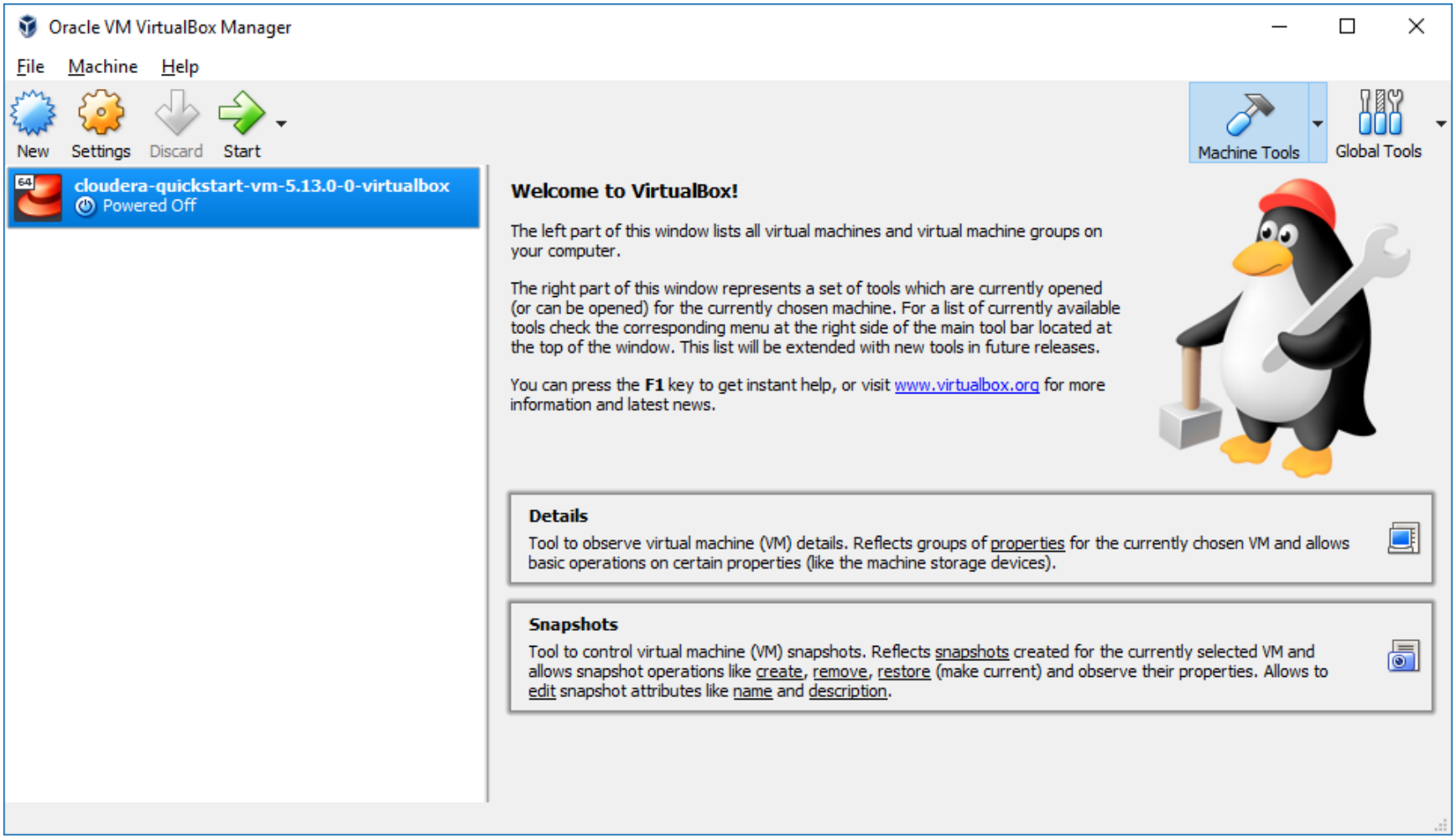
Setting Up the VM



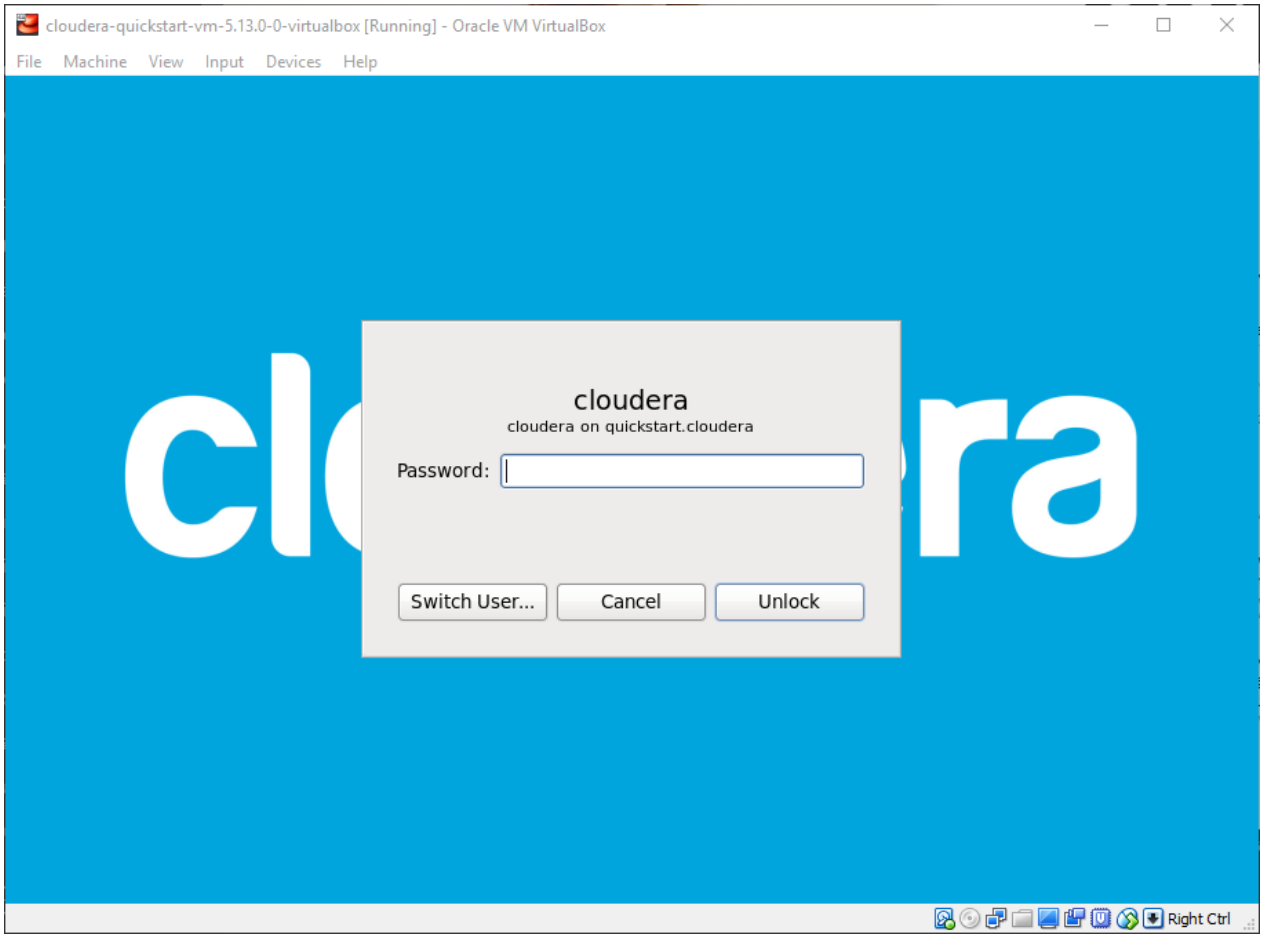
Setting Up the VM



Launch Cloudera VM

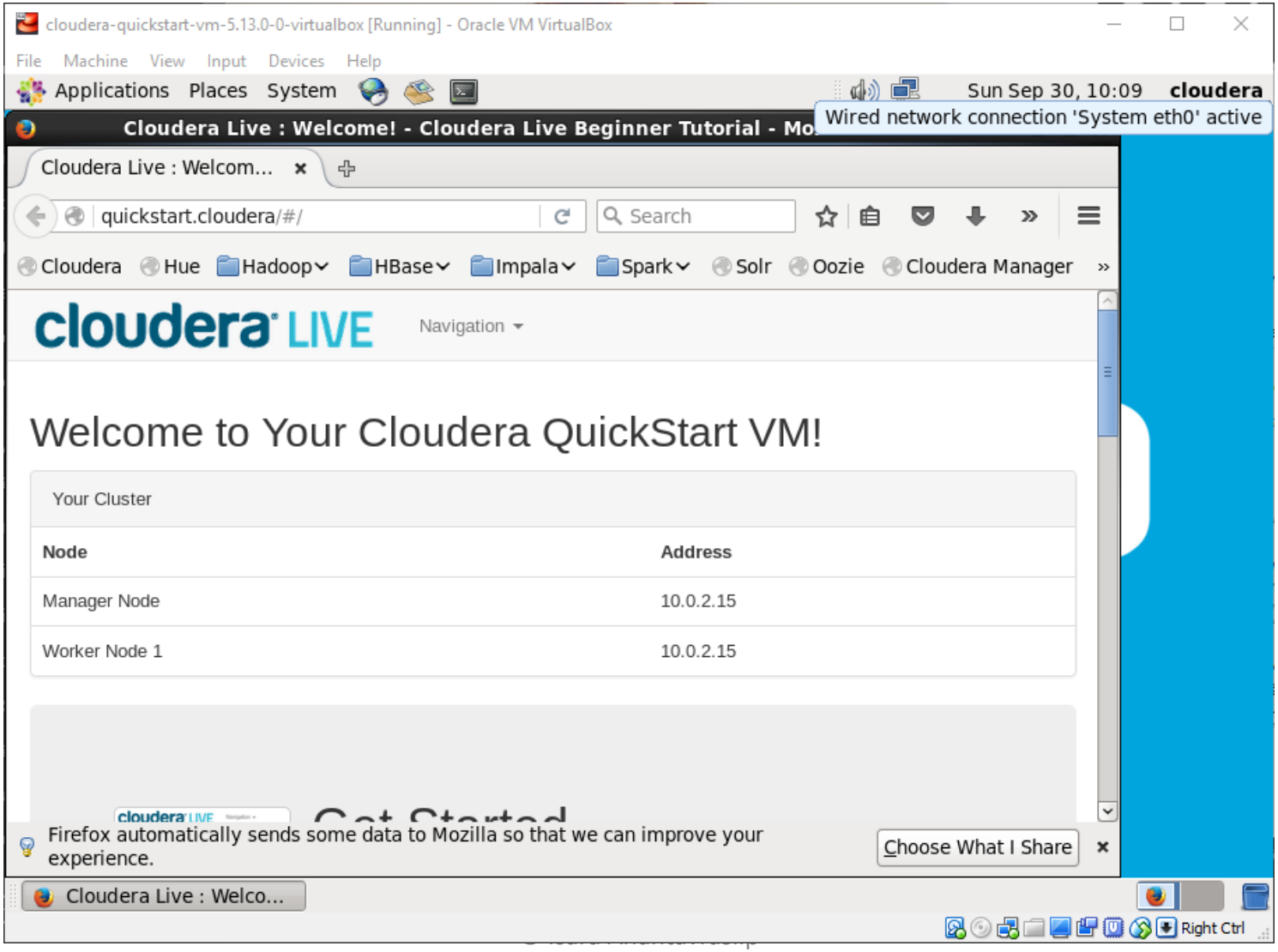


Launch Cloudera VM



Login: cloudera **Password: cloudera**

Launch Cloudera VM



Troubleshooting

- The VM does not start:
  `AMD-V is disabled in the BIOS (or by the host OS) (VERR_SVM_DISABLED).`

  Make sure that your BIOS allows virtualization

- VM freezes when starting:
  It does not freeze, just wait until it finishes loading

Let’s check if we can run Hadoop

- Open terminal

- Type in the following command

```
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar
```

- It should list available commands

Word Count

- Now let's try
  `hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar` **`wordcount`**
- Result
  `Usage: wordcount <in> [<in>...] <out>`
  `[cloudera@quickstart ~]$`
- This is word-counting example
- Let's count some words

Word Files

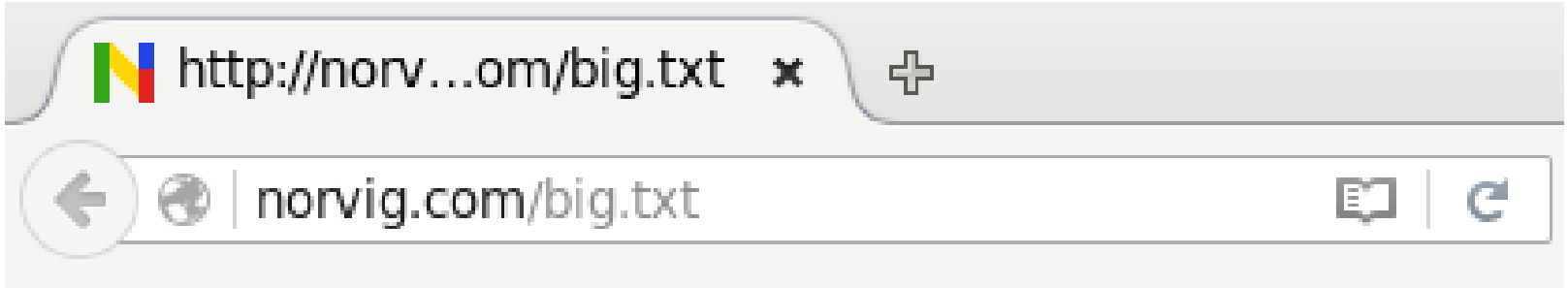
- The Complete Works of William Shakespeare https://ocw.mit.edu/ans7870/6/6.006/s08/lecturenotes/files/t8.shakespeare.txt

- The Project Gutenberg EBook of The Adventures of Sherlock Holmes http://norvig.com/big.txt

Download and Save

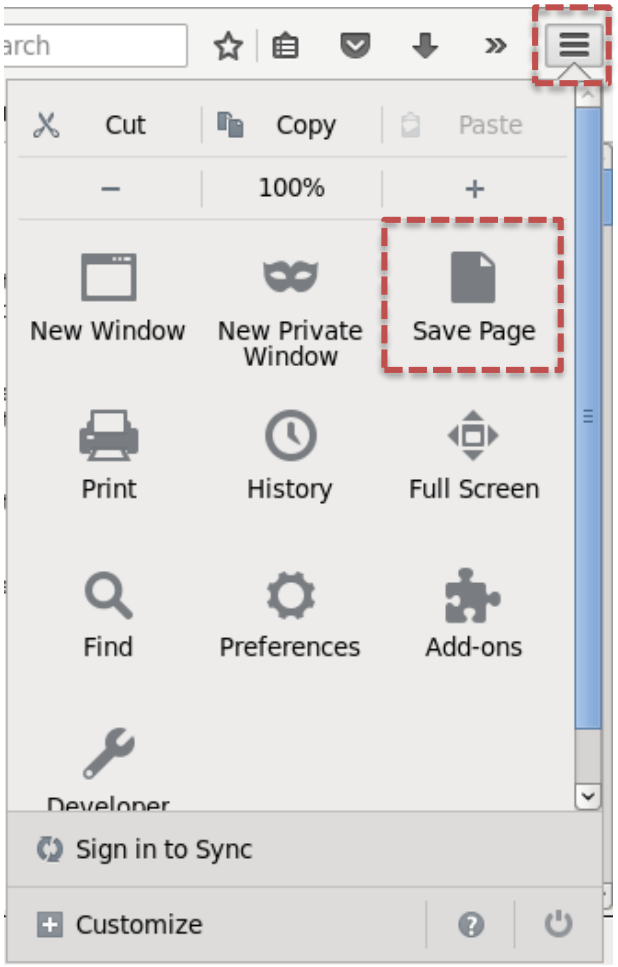
- Open web browser

- Type in or paste the URL



Download and Save

- After the page is loaded, save the file
- Default destination is ~/Download



Let's count the words

- Open terminal and type
  `hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar` **`wordcount big.txt out`**

- It will fail
  `InvalidInputException: Input path does not exist:`

- This is because the file is not yet in HDFS!

Local File System and HDFS

- Hadoop does not store everything in HDFS
- Map results are normally stored in nodes' local file systems
  - Map results are intermediate results which will be sent to reduce task later
  - They do not need redundancy provided by HDFS
  - If a map node fails, Hadoop task manager simply resend the task to another node
- Hadoop HDFS stores
  - Input data: We must put our data into HDFS first
  - Reduce output data: Result of the entire process

Copy the data into HDFS

- Open the terminal and go to Downloads directory
  `cd Downloads/`

- List the files with `ls` or `ls -al`

- You should see your downloaded files
  ```
  [cloudera@quickstart Downloads]$ ls
  big.txt  t8.shakespeare.txt
  ```

Copy the data into HDFS

- Copy the file from local file system to HDFS
  `hadoop fs -copyFromLocal big.txt`

Command:
File system commands

Command Option:
Copy file from local FS to HDFS

- Check whether the file is copied correctly
  `hadoop fs -ls`

- Now, let's try to copy `big.txt` to HDFS again

Other HDFS Command Options

- List the files in current directory
  `hadoop fs -ls`

- Copy files within HDFS
  `hadoop fs -cp big.txt big2.txt`

- Copy files back to local file system
  `hadoop fs -copyToLocal big2.txt`

- Remove files in HDFS
  `hadoop fs -rm big2.txt`

- Show all command options
  `hadoop fs`

Let's count the words (again)

- Open terminal and type
  `hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar` **`wordcount big.txt out`**
- This time it should run
- While it is running, Hadoop will show progress including completed map and reduce tasks

```
18/10/01 13:41:20 INFO mapreduce.Job:  map 0% reduce 0%
18/10/01 13:41:26 INFO mapreduce.Job:  map 100% reduce 0%
18/10/01 13:41:32 INFO mapreduce.Job:  map 100% reduce 100%
```

Copy the result to local FS

- The output is stored in **directory** `out` in HDFS
- You can list the contents inside the directory with:
  `hadoop fs -ls out`
- Then copy the result file back with
  `hadoop fs -copyToLocal out/part-r-00000`
- Now see the contents of the result:
  `more part-r-00000`

What have we done so far?

- We copied files to and from HDFS
- We have run some HDFS file commands
- We have executed MapReduce program
  - The data to be operated is on HDFS
  - But the program is on the local file system
  - WordCount is written in Java but it can be any language

Prepare Compiling Environment

- Most of environment parameters are already set in Cloudera QuickStart, to check type:
  `printenv`

- The following environment should be there:
  JAVA_HOME=/usr/java/jdk1.7.0_67-cloudera
  PATH=/usr/java/jdk1.7.0_67-cloudera/bin

- What we have to do is to set is
  `export`
  HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar

Compiling Word Count

- To compile:
  `hadoop com.sun.tools.javac.Main WordCount.java`
- The result will be multiple class files

- We have to pack them into one JAR file
  `jar cf wc.jar WordCount*.class`
- Result will be a JAR file: `wc.jar`

Running Word Count

- Counting word in the `big.txt` file
  `hadoop jar wc.jar WordCount big.txt out2`
- You should have the same result as previous example
- The result is stored in `out2` directory
- Let's copy to local file system
  `hadoop fs -copyToLocal out2`

Hadoop Jobs

- Hadoop MapReduce process is categorized as a **job**
- A job consists of **tasks**
  - Map tasks
  - Reduce tasks
  - Tasks are scheduled by YARN
  - If a task fails, it will be automatically re-scheduled in another node

Input Splits

- MapReduce separates entire data into smaller chunks or **splits** and feed into map tasks (and later to reduce tasks)
- Splits allow the tasks to be distributed among nodes
- Best size of each splits is the size of a HDFS block
  - Too small, too much scheduling overhead
  - Too large, one split is separated into many nodes
- Hadoop tries to assign map task to the node where the data already resides
  - **locality optimization**

Distributed and Combining Tasks

- A job is split into tasks and tasks are distributed to map nodes
  - Tasks are processed in parallel
- When map tasks are done, the results will be sent to reducer(s)
  - There can be more than one reducers
  - Could also be *zero* reducer if the tasks are simple and can be done as map tasks
- If there are more than one reducers, the map tasks must **partition** the outputs
  - Partition (divide) the outputs into different keys
  - Send different keys to different reducers