

1 Set up a Hadoop cluster in Docker

If you want to test out Hadoop, or don't currently have access to a big Hadoop cluster network, you can set up a Hadoop cluster on your own computer, using Docker. Docker is a popular independent software container platform that allows you to build and ship your applications, along with all its environments, libraries and dependencies in containers. The containers are portable, so you can set up the exact same system on another machine by running some simple Docker commands. Thanks to Docker, it's easy to build, share and run your application anywhere, without having to depend on the current operating system configuration.

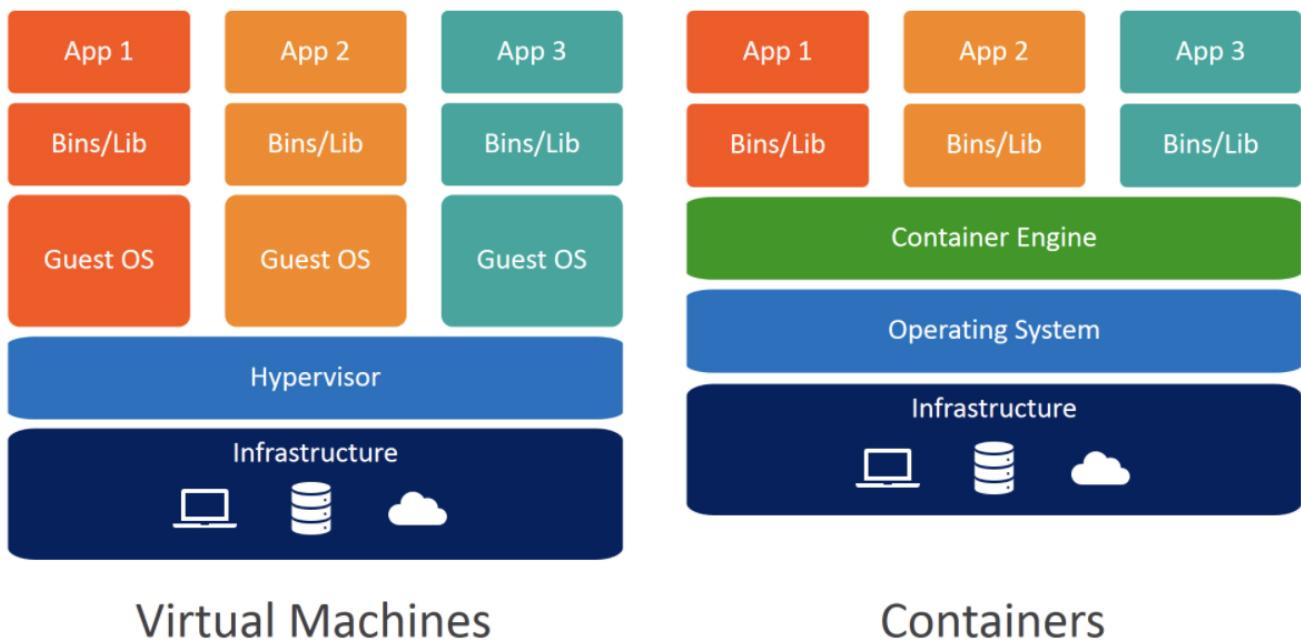
For example, if you have a laptop that is running Windows but need to set up an application that only runs on Linux, thanks to Docker, you don't need to install a new OS or set up a virtual machine. You can set up a Docker container containing all the libraries you need and delete it the moment you are done with your work.

In this practice lesson, we will set up a 3-node Hadoop cluster using Docker and run the classic Hadoop Word Count program to test the system.

1.1 Introduction to Docker

Docker, an open-source project launched in 2013, has helped popularize technology and has helped drive the trend towards containerization and micro-services in software development that's known as cloud-native development.

1.1.1 Understand Virtualization and Containerization



Virtualization

Virtualization is the technique of importing a Guest operating system on top of a Host operating system. This technique was a revelation at the beginning because it allowed developers to run many operating systems in different virtual machines, all running on the same host. These eliminated the need for extra hardware resources.

Containerization

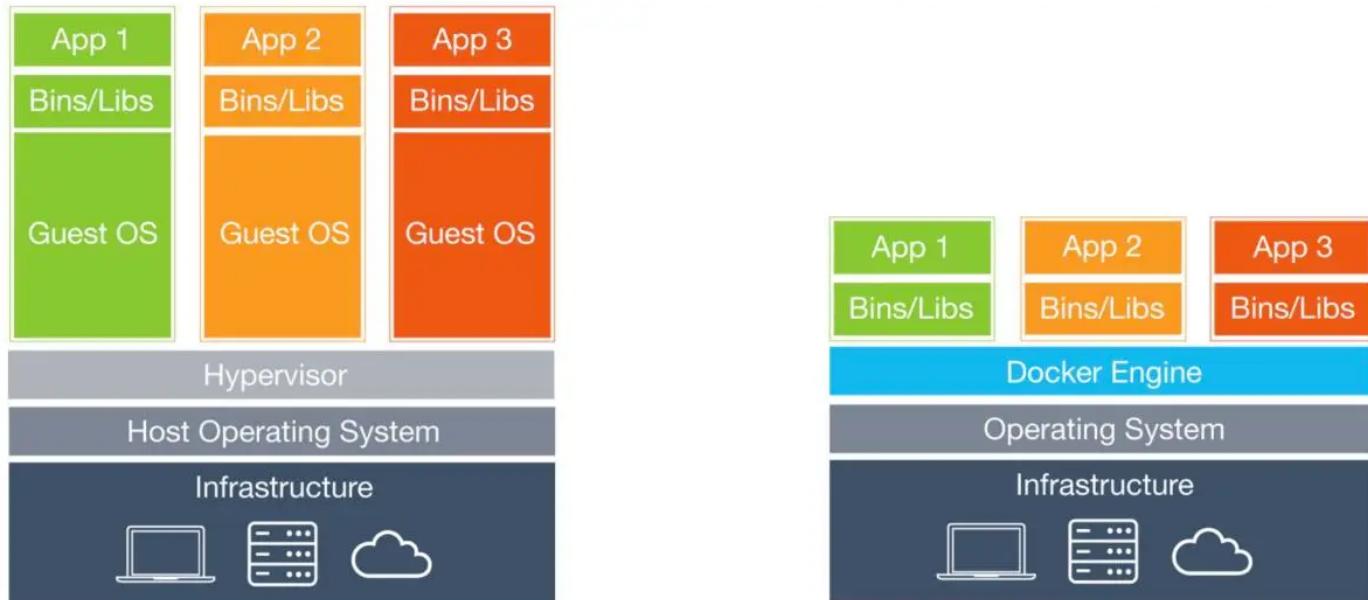
Containerization is a form of operating system virtualization, through which it runs applications in secluded user spaces called containers, all using the same shared operating system (OS).

Differences

Virtualization supports executing multiple operating systems on a single physical server, whereas Containerization supports deploying multiple applications developed in the environment of one operating system residing on a single virtual machine or a server.

Key Factor	Virtualization	Containerization
Technology	One physical machine has multiple OSs residing on it and appears as multiple machines.	The application developed in a host environment with same OS and the same machine executes flawlessly on multiple different environments.
Start-up Time	Higher than containers	Less
Speed of working	VMs being a virtual copy of the host server on its own operating system, VMs are resource-heavy, hence slower.	Containers are faster.
Size	Larger	Smaller
Component that Virtualizes and the one being virtualized	Hypervisors virtualize the underlying hardware (use of the same hardware).	Containers virtualize the operating system (use of the same OS).
Cost of implementation	Higher	Lower
Benefits for	IT enterprise businesses	Software developers and in turn IT businesses

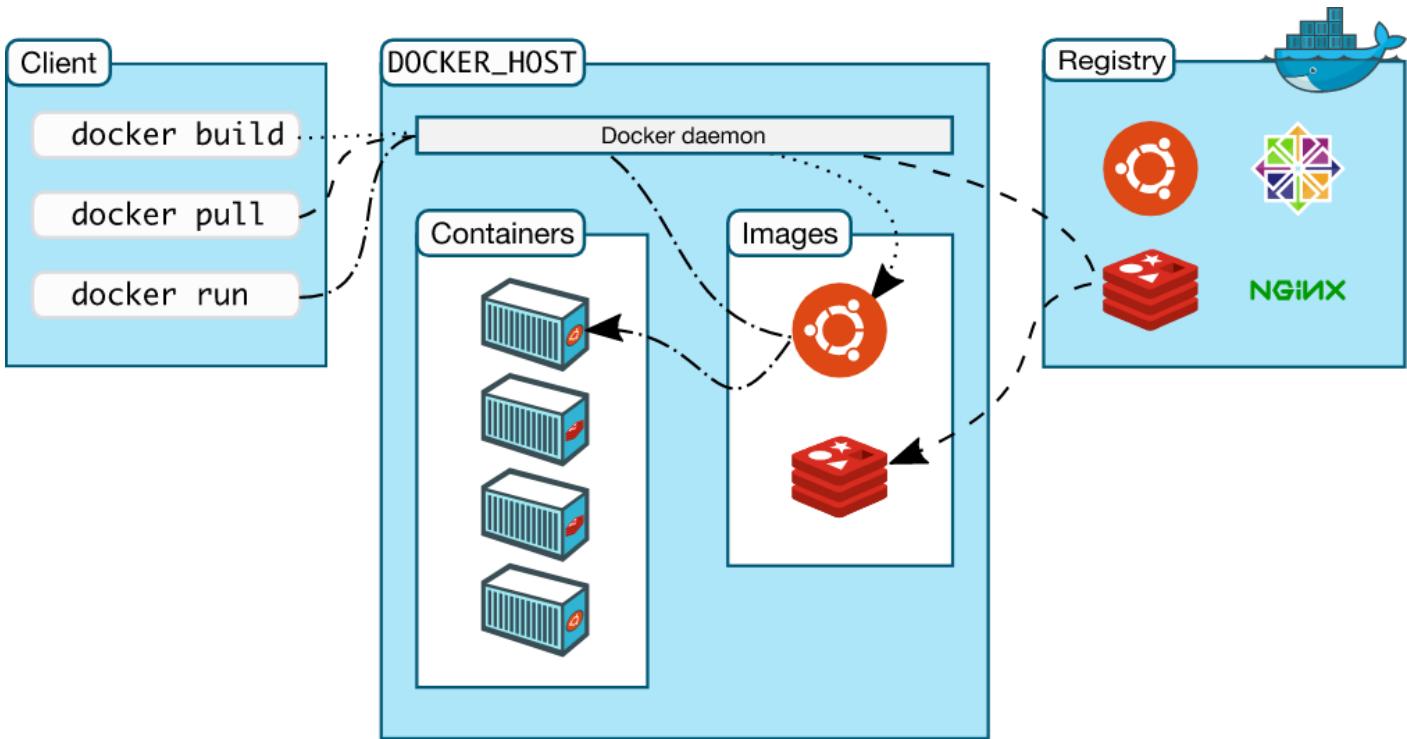
1.1.2 What is Docker?



Docker is an open-source container solution that allows you to containerize and package an application and its dependencies in Docker containers for development and deployment purposes.

A Docker container is a pre-configured environment that includes all of the necessary installations for the application running inside it. Each container has a running application, and each container runs on the Docker Engine, which in turn runs on top of the host operating system.

Docker containers use Docker container images to containerize an application. For every application, there is an official Docker container image that you can use to containerize your application dependencies and executables.



Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package. Initially, built for Linux, Docker now runs on Windows and macOS. To understand how Docker works, let's look at some components you would use to create Docker-containerized applications.

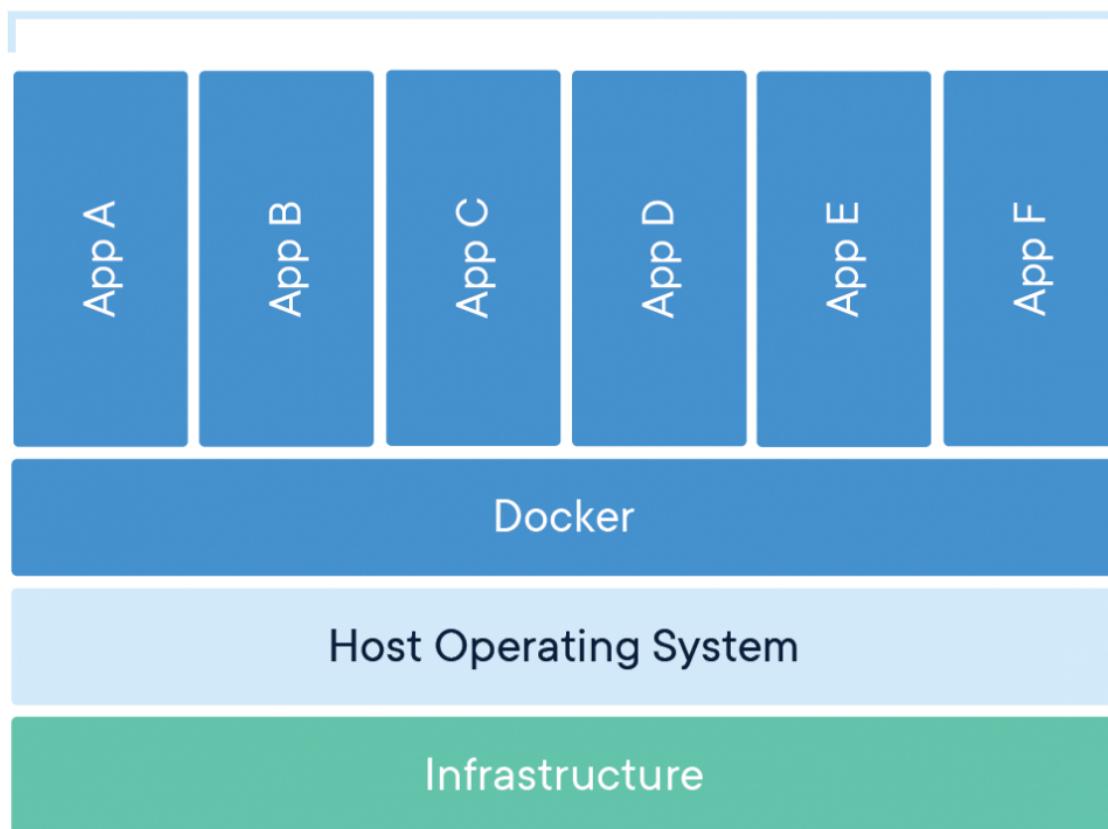
Terms and tool

- **DockerFile:** A DockerFile is a text file that contains instructions on how to build a docker image. A Dockerfile specifies the operating system that will underlie the container, along with the languages, environmental variables, file locations, network ports, and other components it needs—and what the container will do once we run it.
- **Docker Images:** Docker images contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.
- **Docker Container:** A Docker container image is a lightweight, standalone, executable package of software that has everything you need to run an application – code, runtime, system tools, system libraries, and settings.
- **Docker Hub:** Docker Hub is the public repository of Docker images that calls itself the “world’s largest library and community for container images.” It holds over 100,000 container images sourced from commercial software vendors, open-source projects, and individual developers. It includes images that have been produced by Docker, Inc., certified images belonging to the Docker Trusted Registry, and many thousands of other images. All Docker Hub users can share their images at will. They can also download predefined base images to use as a starting point for any containerization project.
- **Docker Daemon:** Docker Daemon is the background service running on the host that manages the building, running, and distributing Docker containers. The daemon is the process that runs in the operating system in which clients speak.
- **Docker Engine:** Docker Engine is a client-server application that supports the tasks and workflows involved to build, ship, and run container-based applications. The engine creates a server-side daemon process that hosts images, containers, networks, and storage volumes.
- **Docker Registry:** The Docker Registry is where the Docker Images are stored. The Registry can be either a user’s local repository or a public repository like a Docker Hub allowing multiple users to

collaborate in building an application. Even with several teams within the same organization can exchange or share containers by uploading them to the Docker Hub, which is a cloud repository similar to GitHub.

- Docker Compose: Docker-compose is for running multiple containers as a single service. It does so by running each container in isolation but allowing the containers to interact with one another.
- Docker Swarm: Docker swarm is a service for containers that allows IT administrators and developers to create and manage a cluster of swarm nodes within the Docker platform. Each node of Docker swarm is a Docker daemon, and all Docker daemons interact using the Docker API. A swarm consists of two types of nodes: a manager node and a worker node. A manager node maintains cluster management tasks. Worker nodes receive and execute tasks from the manager node.

Containerized Applications



1.1.3 Applications of Docker

Docker is an excellent tool for developers and system administrators. We can use it in multiple stages of the DevOps cycle and for the rapid deployment of applications. It allows the developers to build an application and package an application with all its dependencies into a Docker run container that can run in any environment.

Docker allows us to develop an application and its supporting components efficiently using the containers. These containers are lightweight and can run directly within the host machine's kernel. Thus, it allows running more containers on a single hardware.

It provides a loosely isolated environment that is secure enough to run multiple containers simultaneously on a particular host.

Any unforeseen condition or situation can halt the software development lifecycle and affect the business organization significantly. But, with Docker, it can be mitigated. Docker allows the functionality to easily replicate the file or Docker image to new hardware and retrieve it later in case of any issues. In case of rollback of any particular feature or version, Docker can be useful to revert to the last version of the Docker image quickly.

Docker allows developing, testing, and deploying applications faster. A software development life cycle is long, as it includes testing, making necessary changes, finding bugs, and deploying it to see the final results. Docker allows the developers to find bugs in the initial stages of development so that they can be fixed in the development environment and can be redeployed for testing and validation.

1.1.4 Set up Docker

To check the version of your Docker Engine, Machine and Compose, use the following commands

```
docker --version
docker-compose --version
docker-machine --version
```

If it's your first time running Docker, test to make sure things are working properly by launching your first Dockerized web server.

```
docker run -d -p 80:80 --name myserver nginx
```

1.2 Hortonworks Sandbox on Docker

- Download latest [scripts](#) from Hortonworks Data Platform (HDP) for Docker
- Make sure Docker is installed, `cd` to your unzipped folder, run `sh`

```
sh docker-deploy-{HDPversion}.sh
docker start sandbox-hdp sandbox-proxy
docker ps
```

- Output will look something like

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
98c5d2e6ace7	hortonworks/sandbox-proxy:1.0	"nginx -g 'daemon of..."	2 hours ago
Up 6 minutes	0.0.0.0:1080->1080/tcp,...	sandbox-proxy	
02b390f86aa1	hortonworks/sandbox-hdp:3.0.1	"/usr/sbin/init"	2 hours ago
Up 6 minutes	22/tcp, 4200/tcp, 8080/	sandbox-hdp	

1.3 Generate Hadoop Docker image

To install Hadoop in a Docker container, we need a Hadoop Docker image. To generate the image, we will use the Big Data Europe repository. Start by cloning this docker-Hadoop repository from Github

```
git clone https://github.com/big-data-europe/docker-hadoop.git
```

The sample repository above has a Hadoop docker-compose.yml set and ready to be deployed to Docker containers. Navigate to the cloned folder, and then run the following command to start the container using docker-compose

```
docker-compose up -d
```

The docker-compose up will check the containers set in the docker-compose.yml, download them and run them within the Docker engine.

The -d flag will set the container to run in a detachable model, i.e., in the background. The output would be:

```
Running 28/28
  #: datanode Pulled
    #: 3ca2ec07878c Pull complete
    #: 26c2dd45430e Pull complete
                                         356.8s
                                         346.0s
                                         346.6s
```

```

    #: 13c9c87a46cb Pull complete                                347.3s
  #: nodemanager1 Pulled                                       356.8s
  #: 3192219af04 Pull complete                                180.1s
  #: 7127a1d8cced Pull complete                               240.3s
  #: 883a89599900 Pull complete                               240.4s
  #: 77920a3e82af Pull complete                               240.5s
  #: 92329e81aec4 Pull complete                               348.9s
  #: f373218fec59 Pull complete                               349.2s
  #: aa53513fe997 Pull complete                               349.6s
  #: 8b1800105b98 Pull complete                               349.9s
  #: c3a84a3e49c8 Pull complete                               350.2s
  #: a65640a64a76 Pull complete                               350.6s
  #: beaa171f32f6 Pull complete                               351.0s
  #: 50dda04de8a9 Pull complete                               352.0s
  #: historyserver Pulled                                     356.8s
  #: b2dc88cebe05 Pull complete                               346.0s
  #: 13b908760168 Pull complete                               346.6s
  #: 0991d53828a1 Pull complete                               347.5s
  #: resourcemanager Pulled                                   356.7s
  #: b0d764123f3e Pull complete                               346.0s
  #: b04394ddb35d Pull complete                               346.9s
  #: namenode Pulled                                       356.8s
  #: facffb3a6de3 Pull complete                               346.0s
  #: c71a6df73788 Pull complete                               346.8s
  #: 73b8c0ccb707 Pull complete                               347.6s
[+] Running 9/9
  #: Network docker-hadoop_default                         Created      0.1s
  #: Volume "docker-hadoop_hadoop_namenode"                Created      0.0s
  #: Volume "docker-hadoop_hadoop_datanode"                Created      0.0s
  #: Volume "docker-hadoop_hadoop_historyserver"           Created      0.0s
  #: Container nodemanager                                Started     1.0s
  #: Container namenode                                  Started     0.9s
  #: Container datanode                                  Started     0.7s
  #: Container historyserver                            Started    resourcemanager
Started

```

After everything is done, you can check the running Hadoop containers using the command `docker ps`

CONTAINER ID	IMAGE	COMMAND
ad6328658ac9	bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8	<code>"/entrypoint.sh</code>
<code>/run..."</code>	0.0.0.0:9000->9000/tcp, :::9000->9000/tcp, 0.0.0.0:9870->9870/tcp, :::9870->9870/tcp	
	namenode	
ea0160275f54	bde2020/hadoop-resourcemanager:2.0.0-hadoop3.2.1-java8	<code>"/entrypoint.sh</code>
<code>/run..."</code>	8088/tcp	
	resourcemanager	
c981235e7a5d	bde2020/hadoop-historyserver:2.0.0-hadoop3.2.1-java8	<code>"/entrypoint.sh</code>
<code>/run..."</code>	8188/tcp	
	historyserver	
ccc813cd6312	bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8	<code>"/entrypoint.sh</code>
<code>/run..."</code>	9864/tcp	
	datanode	
686c7604243c	bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8	<code>"/entrypoint.sh</code>
<code>/run..."</code>	8042/tcp	
	nodemanager	

1.4 Testing Hadoop Cluster

Go to <http://localhost:9870> to view the current status of the system from the namenode

Enter into the running namenode container

```
docker exec -it namenode bash
```

Create some example text files to work with

```
mkdir input
cd input
```

```
echo "apple banana" > fruit.txt
echo "cat dog" > pet.txt
```

Create input_hdfs directory on HDFS, put created files to all datanodes

```
hdfs dfs -mkdir -p input_hdfs
hdfs dfs -put * input_hdfs
```

Then you can check your files on <http://localhost:9870/explorer.html> or

```
hdfs dfs -ls input_hdfs
```

1.5 Run MapReduce Job

Go to your host terminal

```
wget https://repo1.maven.org/maven2/org/apache/hadoop/hadoop-mapreduce-
examples/2.7.1/hadoop-mapreduce-examples-2.7.1-sources.jar
docker cp hadoop-mapreduce-examples-2.7.1-sources.jar namenode:/
```

Back to namenode terminal

```
cd /
hadoop jar hadoop-mapreduce-examples-2.7.1-sources.jar
org.apache.hadoop.examples.WordCount input_hdfs output_hdfs
```

From above:

- org.apache.hadoop.examples is the class.
- WordCount is the function.
- input_hdfs is the directory where we have our file.
- output_hdfs where the files will be generated.

Show job output

- Check output files

```
hdfs dfs -ls output_hdfs
```

- Show output

```
hdfs dfs -cat output_hdfs/part-r-00000
```

- Output from above example will look like

```
apple 1
banana 1
cat 1
dog 1
```

1.6 Test with a sample text file

From host

```
wget https://raw.githubusercontent.com/apache/hadoop/trunk/LICENSE.txt
docker cp LICENSE.txt namenode:/
```

From namenode

```
hdfs dfs -mkdir -p input_test
hdfs dfs -put LICENSE.txt input_test
hadoop jar hadoop-mapreduce-examples-2.7.1-sources.jar
org.apache.hadoop.examples.WordCount input_test output_test
hdfs dfs -cat output_test/part-r-00000
```

1.7 Shut down your containers

To safely shut down the cluster and remove containers

```
docker-compose down
```

2 Common usage of hdfs dfs ...

2.1 List Files

- `-ls /`
 - List all files/directories for given hdfs destination path.
- `-ls -d /hadoop`
 - Directories are listed as plain files. List details of hadoop folder.
- `-ls -h /data`
 - Format file sizes in a human-readable fashion
- `-ls -R /hadoop`
 - Recursively list all files in hadoop directory & all subdirectories in hadoop directory.
- `-ls /hadoop/dat*`
 - List all files matching pattern. (starts with `\text{quot}single dat\text{quot}single`)

2.2 Read/Write Files

- `-text /hadoop/derby.log`
 - Take a source file and outputs in text format
- `-cat /hadoop/test`
 - Display content of HDFS file test on your stdout
- `-appendToFile /home/ubuntu/test1 /hadoop/text2`
 - Appends content of a local file test1 to a hdfs file test2

2.3 Upload/Download Files

- `-put /home/sample /hadoop`
 - Copies file from local to HDFS.
- `-put -f /home/sample /hadoop`
 - Copies file from local to HDFS, if local already exists in given destination path, overwrite it.
- `-put -l /home/sample /hadoop`
 - Copies file from local to HDFS. Allow DataNode to lazily persist file to disk. Forces replication factor of 1.
- `-put -p /home/sample /hadoop`
 - Copies file from local to HDFS. Preserves access and modification times, ownership and mode.
- `-get /newfile /home/`
 - Copies file from HDFS to local.

- `-get -p /newfile /home/`
 - Copies file from HDFS to local. Preserves access and modification times, ownership and mode.
- `-get /hadoop/*.txt /home/`
 - Copies all files matching pattern from local to HDFS.
- `-copyFromLocal /home/sample /hadoop`
 - Similar to put command, except that source is restricted to a local file reference.
- `-copyToLocal /newfile /home/`
 - Similar to put command, except that destination is restricted to a local file reference.
- `-moveFromLocal /home/sample /hadoop`
 - Similar to put command, except that source is deleted after it\text{single} s copied.

2.4 File Management

- `-cp /hadoop/file1 /hadoop1`
 - Copies file from source to destination on HDFS. file1 from hadoop directory to hadoop1 directory.
- `-cp -p /hadoop/file1 /hadoop1`
 - Preserves access & modification times, ownership & mode.
- `-cp -f /hadoop/file1 /hadoop1`
 - Overwrites destination if it already exists.
- `-mv /hadoop/file1 /hadoop1`
 - Move files that match specified file pattern `<src>` to a destination `<dst>`. When moving multiple files, destination must be a directory.
- `-rm /hadoop/file1`
 - Deletes file (sends it to trash).
- `-rm -r /hadoop`
- `-rm -R /hadoop`
- `-rmdir /hadoop`
 - Deletes directory & any content under it recursively.
- `-rm -skipTrash /hadoop`
 - bypass trash, if enabled, delete specified file(s) immediately.
- `-rm -f /hadoop`
 - If file does not exist, do not display a diagnostic message or modify exit status to reflect an error.
- `-rmdir /hadoop1`
 - Delete a directory.
- `-mkdir /hadoop2`

- Create a directory in specified HDFS location.
- **-mkdir -f /hadoop2**
 - Not fail even if directory already exists.
- **-touchz /hadoop3**
 - Creates a file of zero length at <path> with current time as timestamp of that <path>.

2.5 Ownership and Validation

- **-checksum /hadoop/file1**
 - Dump checksum information for files that match file pattern <src> to stdout.
- **-chmod 755 /hadoop/file1**
 - Changes permissions of file.
- **-chmod -R 755 /hadoop**
 - Changes permissions of files recursively.
- **-chown owner:group /hadoop**
 - Changes owner of file
- **-chown -R owner:group /hadoop**
 - Changes owner of files recursively.
- **-chgrp my_group /hadoop**
 - Changes group association of file.
- **-chgrp -R my_group /hadoop**
 - Changes group association of files recursively.

2.6 Filesystem

- **-df /hadoop**
 - Show capacity, free and used space of filesystem.
- **-df -h /hadoop**
 - Show capacity, free and used space of filesystem, formats sizes of files in a human-readable fashion.
- **-du /hadoop/file**
 - Show amount of space, in bytes, used by files that match specified file pattern.
- **-du -s /hadoop/file**
 - Rather than showing size of each individual file that matches pattern, show total (summary) size.
- **-du -h /hadoop/file**
 - Show amount of space, in bytes, used by files that match specified file pattern. Formats sizes of files in a human-readable fashion.

2.7 Administration

- `hdfs balancer -threshold 30`
 - Runs a cluster balancing utility. Percentage of disk capacity. Overwrite default threshold.
- `hadoop version`
 - Check version of Hadoop.
- `hdfs fsck /`
 - Check health of Hadoop file system.
- `hdfs dfsadmin -safemode leave`
 - Turn off safemode of NameNode.
- `hdfs dfsadmin -refreshNodes`
 - Re-read hosts and exclude files to update set of Datanodes that are allowed to connect to Namenode and those that should be decommissioned or recommissioned.
- `hdfs namenode -format`
 - Formats NameNode.