Getting Started with Hadoop

May 28, 2018

Michael Völske, Shahbaz Syed

Web Technology & Information Systems
Bauhaus-Universität Weimar

What is Hadoop

- Started in 2004 by Yahoo
- Open-Source implementation of Google MapReduce, Google Filesystem and Google BigTable
- Apache Software Foundation top level project
- Written in Java

What is Hadoop

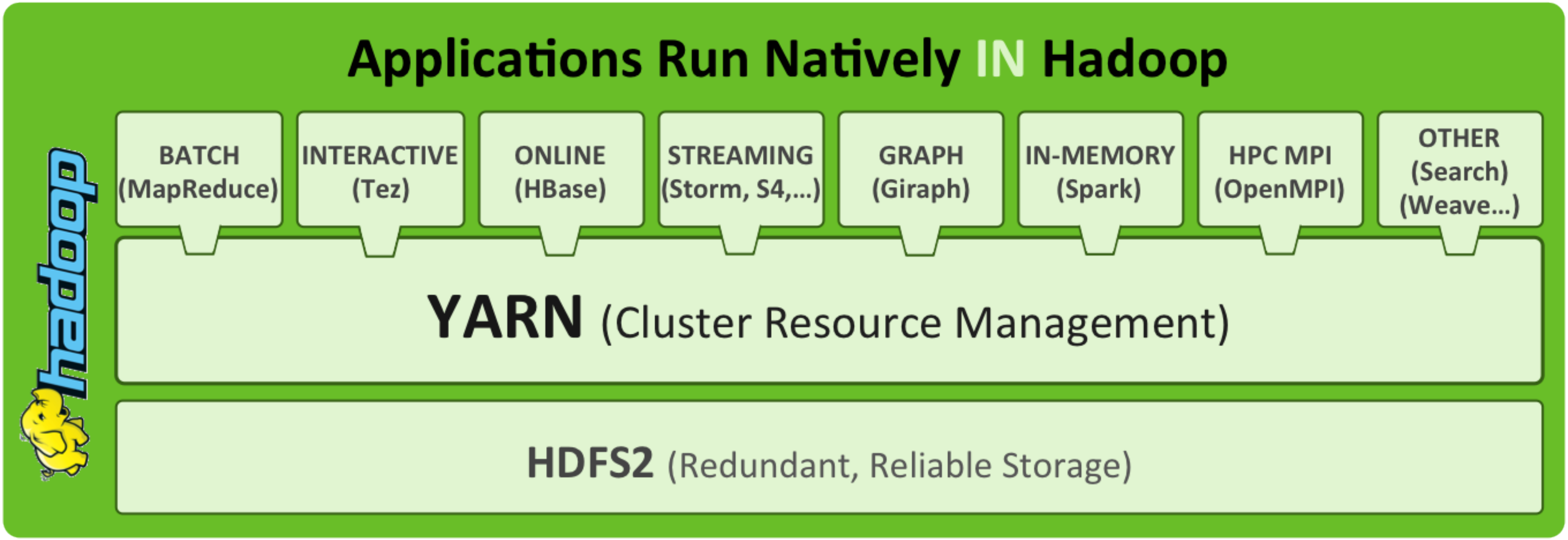
- Scale out, not up!
  - 4000+ nodes, 100PB+ data
  - cheap commodity hardware instead of supercomputers
  - fault-tolerance, redundancy

- Bring the program to the data
  - storage and data processing on the same node
  - local processing (network is the bottleneck)

- Working sequentially instead of random-access
  - optimized for large datasets

- Hide system-level details
  - User doesn’t need to know what code runs on which machine

What is Hadoop


Applications Run Natively IN Hadoop
hadoop
BATCH (MapReduce)
INTERACTIVE (Tez)
ONLINE (HBase)
STREAMING (Storm, S4,...)
GRAPH (Giraph)
IN-MEMORY (Spark)
HPC MPI (OpenMPI)
OTHER (Search) (Weave...)
YARN (Cluster Resource Management)
HDFS2 (Redundant, Reliable Storage)

[http://hortonworks.com/hadoop/yarn/]

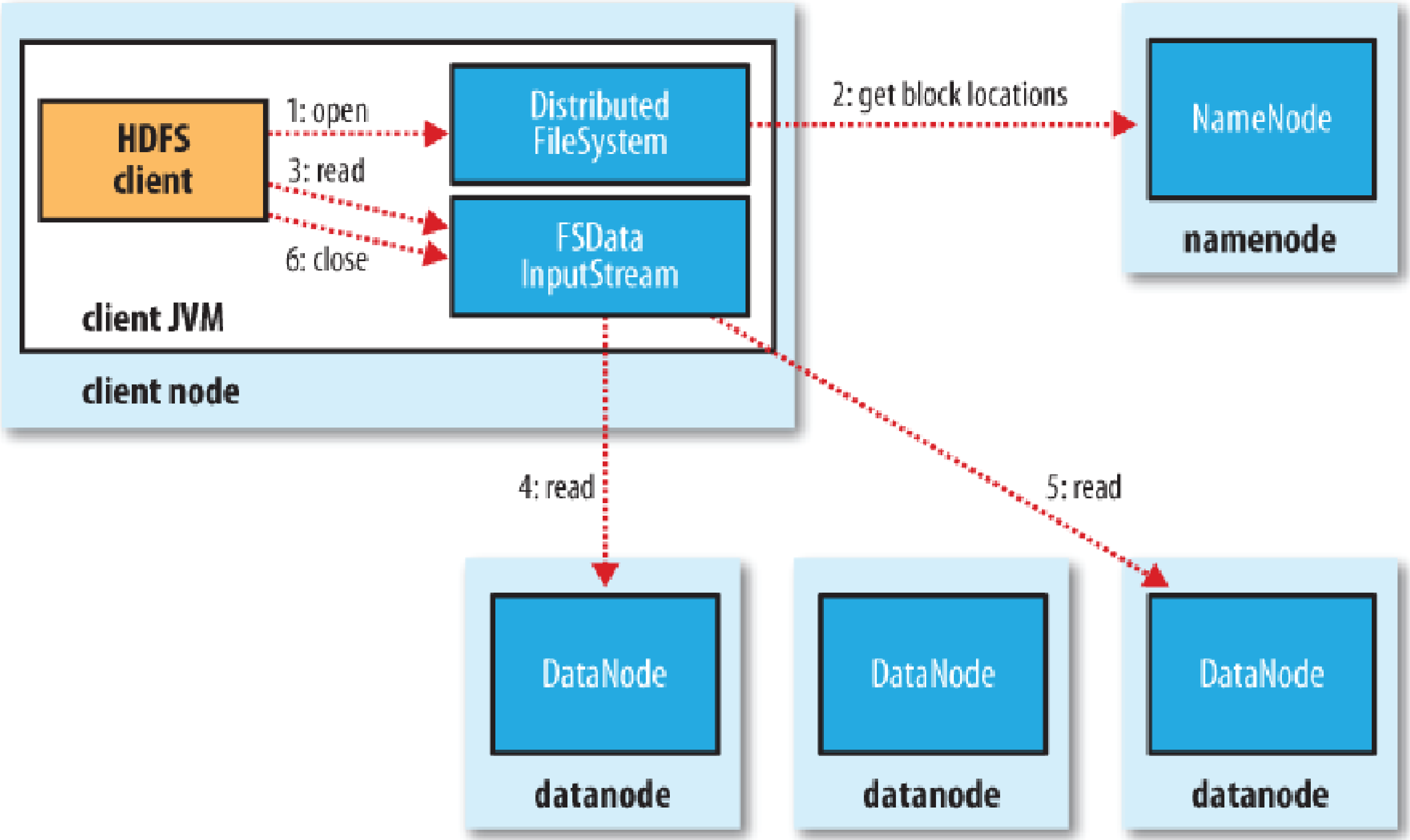
HDFS – Distributed File System

HDFS Overview

- Designed for storing large files
- Files are split in blocks
- Integrity: Blocks are checksummed
- Redundancy: Each block stored on multiple machines
- Optimized for sequentially reading whole blocks
- Daemon processes:
  - NameNode: Central registry of block locations
  - DataNode: Block storage on each node

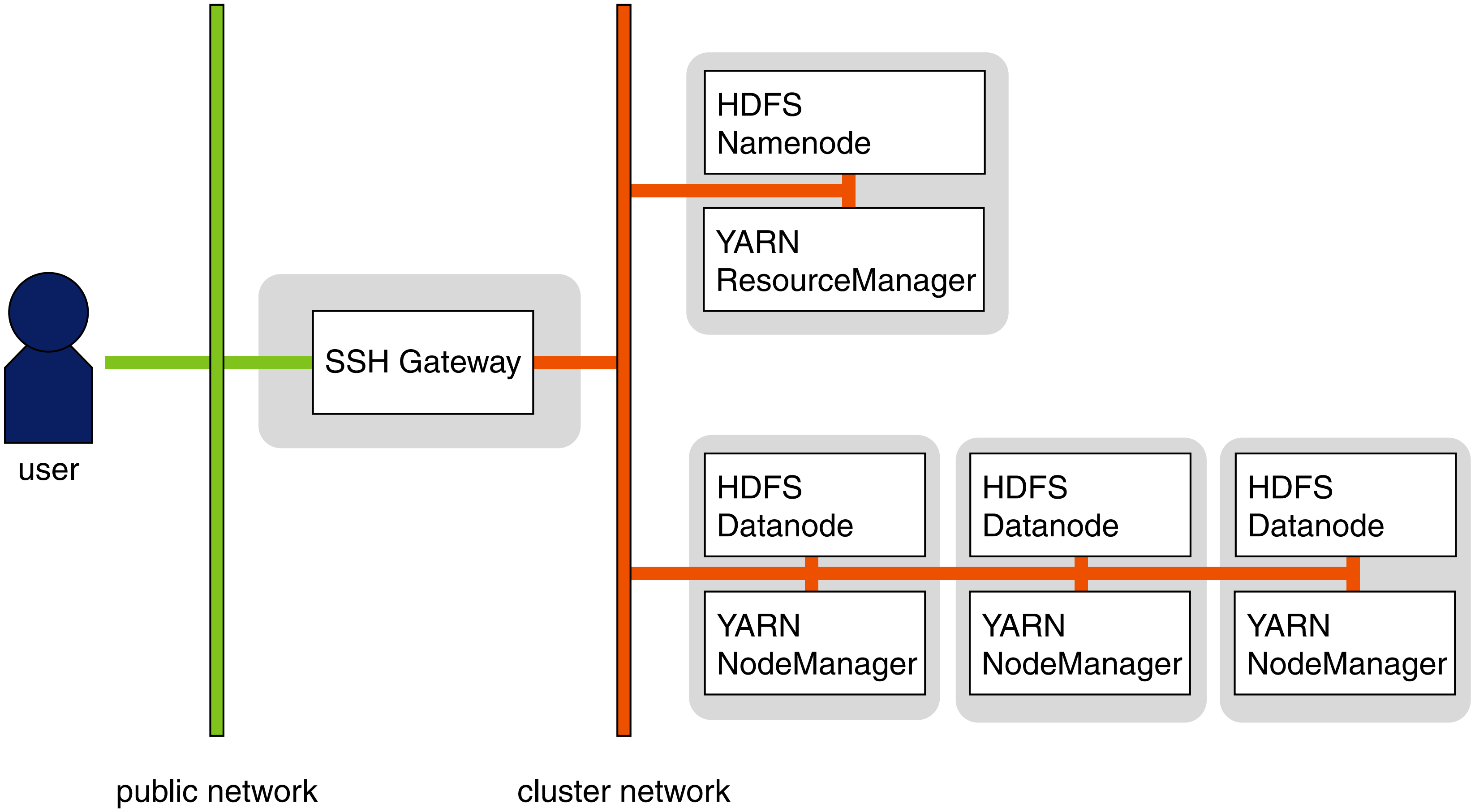
HDFS – Distributed File System

Reading Files



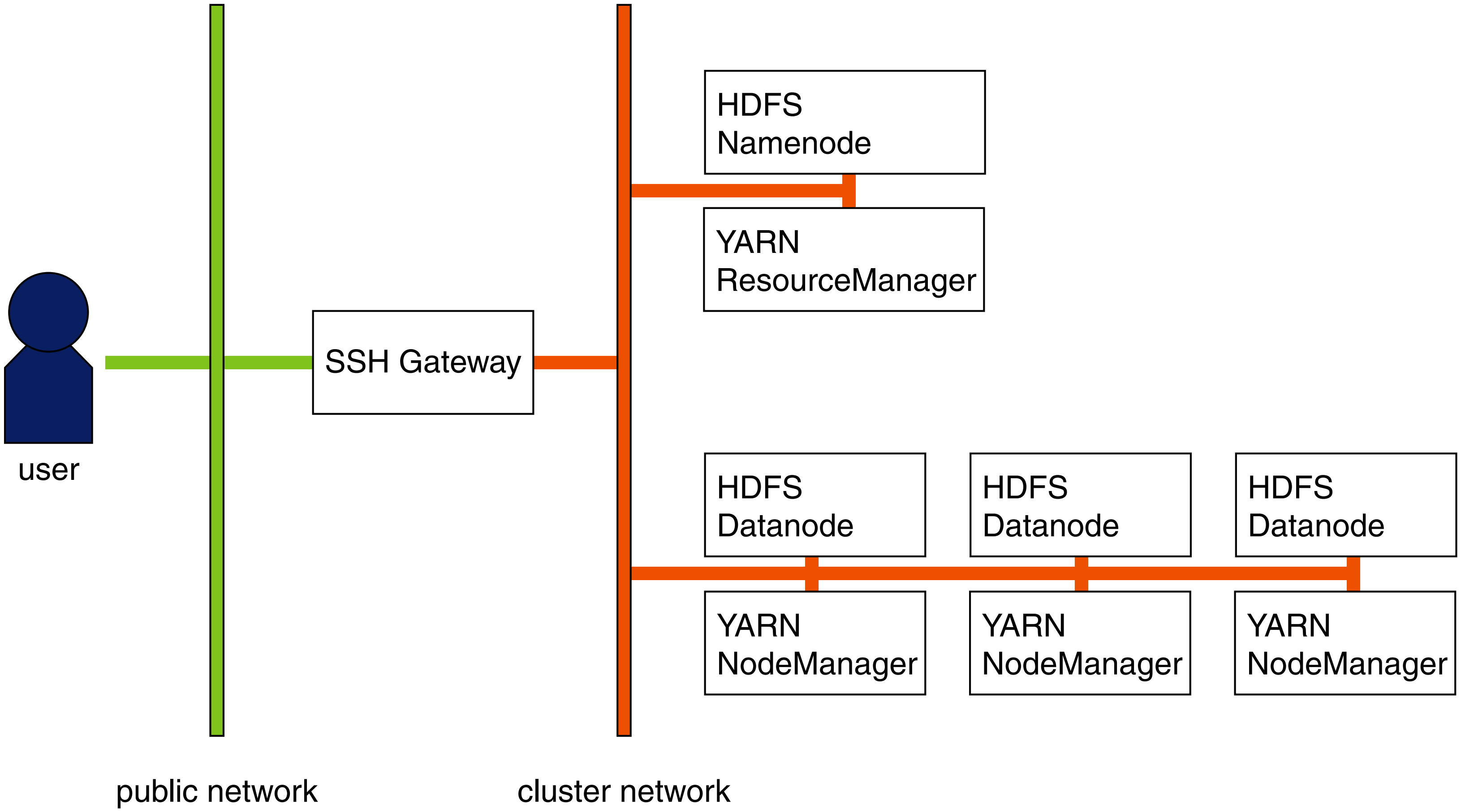
A Virtual Hadoop Cluster

Typical Cluster Network Layout



A Virtual Hadoop Cluster

Simplification: simulate only relevant processes



Docker tutorial

. Introduction to Docker

- ❑ Why Docker?
- ❑ Container vs Virtual Machine
- ❑ Images & Containers
- ❑ Docker workflow
- ❑ Dockerfile

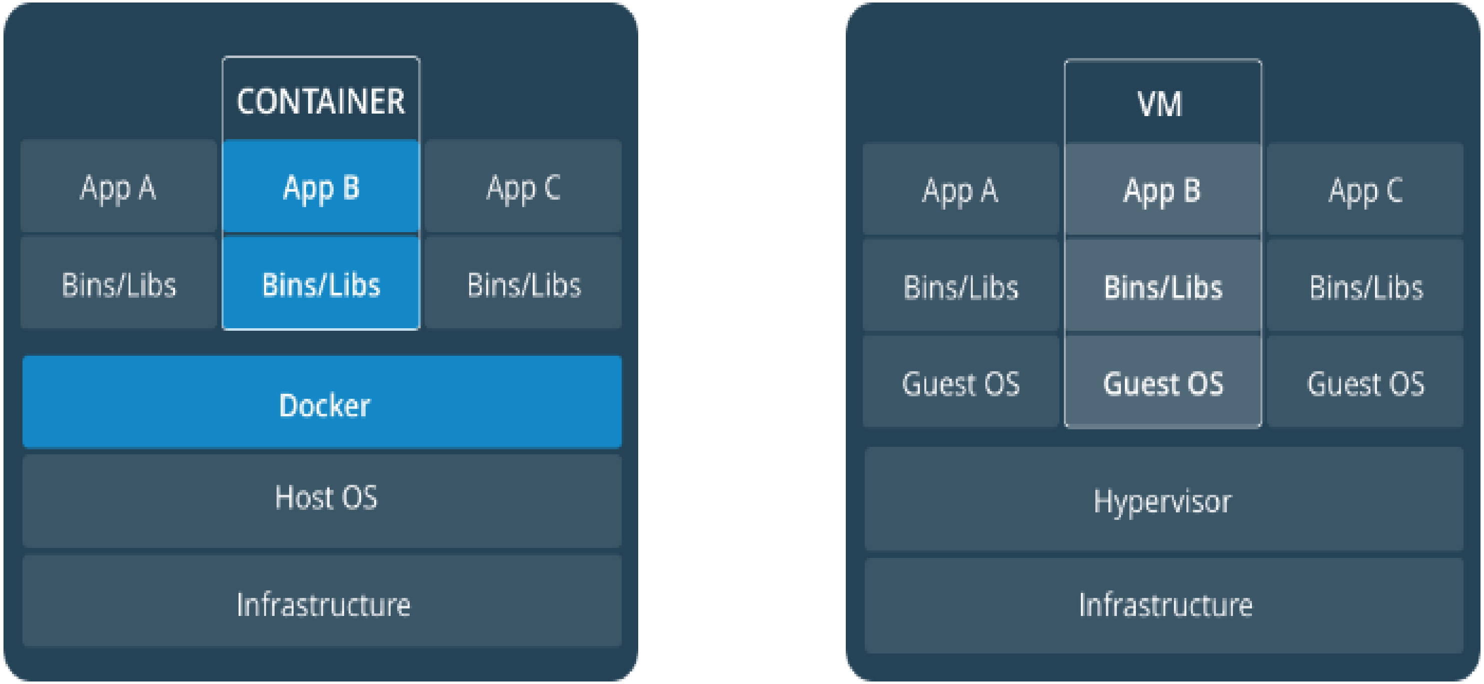
Why Docker?

Docker provides lightweight **containerization** of applications.

- Containerization : Use of Linux **containers** to deploy applications
- Container : Self-contained, lightweight environment
- Run applications with exactly the same behavior on all platforms
- (Un)Install packages/applications without convoluting the host system
- Highly suitable for developing scalable, micro-services based applications (e.g. Netflix)

Container vs Virtual Machine

- A **container** runs **natively** on Linux and **shares** the kernel of the host machine with other containers. (minimal memory consumption)
- A **VM** runs a **full blown guest OS** with **virtual access** to host resources. (extra memory allocated than needed)



[Container vs VM]

Images & Containers

Analogy => {Image : Containers} = {Class : Objects}

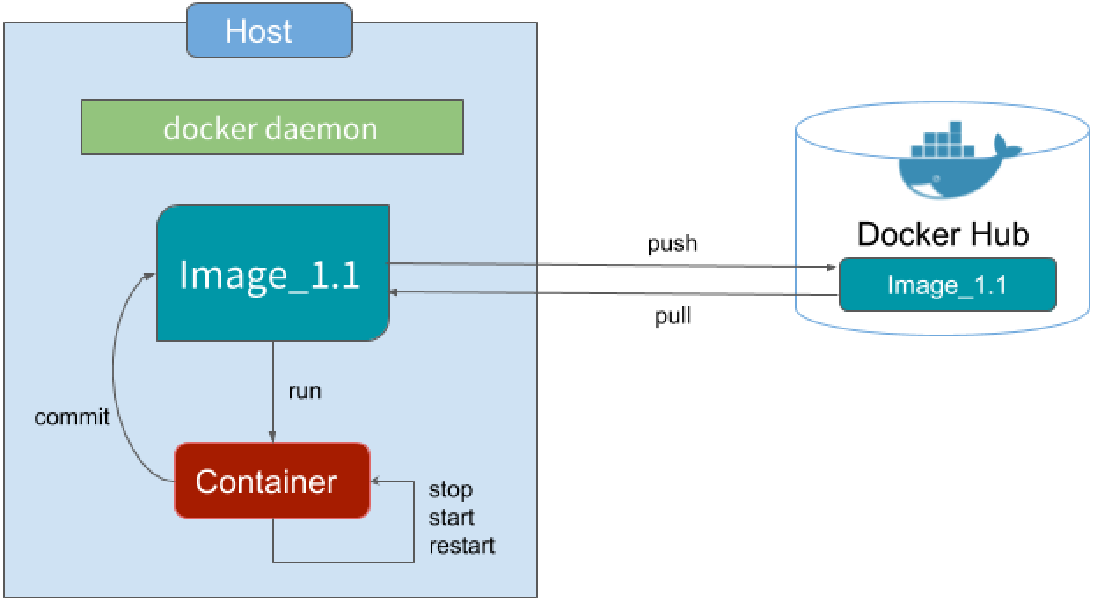
A Docker **Image**

- is an **immutable snapshot** of a filesystem
- contains basic kernel & runtime needed for building larger systems (e.g base ubuntu image)
- is built using Dockerfile (a recipe to build an environment from scratch)

A Docker **Container**

- is a **temporary filesystem** on top of a base **Image**
- saves all installations as a stack of layers
- discards all layers when stopped and removed
- consists of its own network stack (private address) to communicate with the host
- has options to **start**, **stop**, **restart**, **kill**, **pause**, **unpause**

Docker workflow



[local workflow]

Run in terminal: `docker run --rm hello-world`

Docker workflow

Run in Terminal: `docker run --rm -it ubuntu:16.04 bash`

- Try creating some files, then exit the container and start it again
- No persistence by default. How to address this?

Docker workflow

Run in Terminal: `docker run --rm -it ubuntu:16.04 bash`

- ❑ Try creating some files, then exit the container and start it again
- ❑ No persistence by default. How to address this?

Run in Terminal:

```
docker run --rm -it
    -v ./workspace:/my-folder ubuntu:16.04 bash
```

- ❑ Mounting persistent **volumes** to work around this.

Docker workflow

Run in Terminal: `docker run --rm -it ubuntu:16.04 bash`

- ❑ Try creating some files, then exit the container and start it again
- ❑ No persistence by default. How to address this?

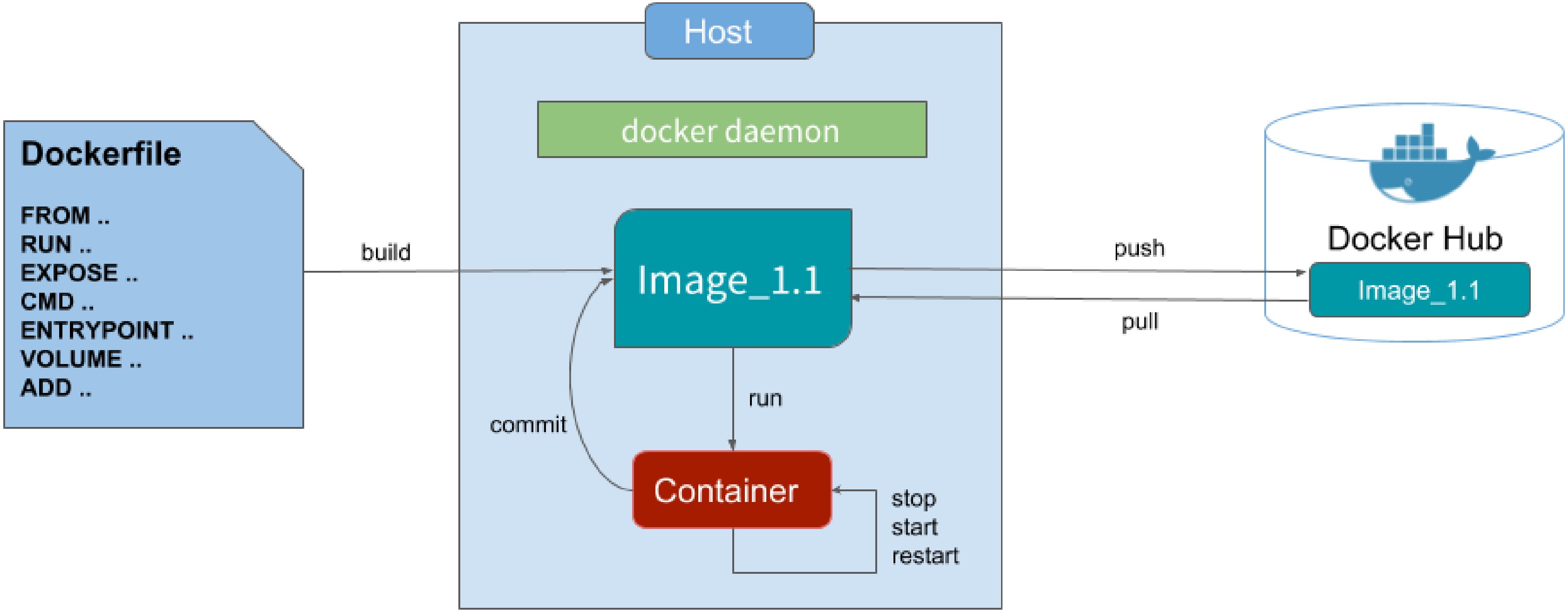
Run in Terminal:

```
docker run --rm -it
    -v ./workspace:/my-folder ubuntu:16.04 bash
```

- ❑ Mounting persistent **volumes** to work around this.

- ❑ Persistence for system changes: build a new image!

Docker workflow



Run in terminal:

```
docker build -t test-image .
docker run --rm -it test-image bash
```

Dockerfile

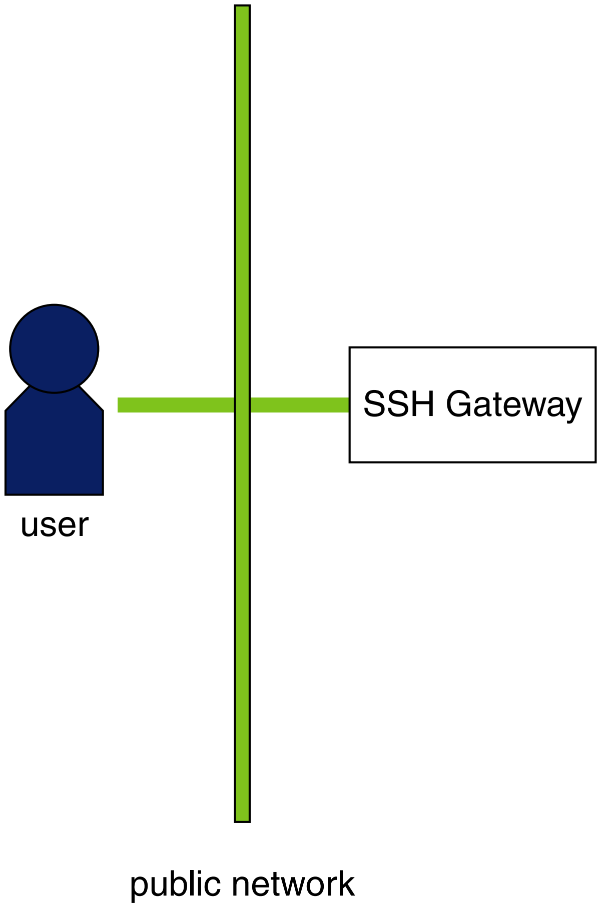
A script which contains a **collection of commands**(docker and Linux) that will be executed **sequentially** in the docker environment for **building** a new docker image.

- FROM : Name/Path of the base image for building a new image; must be the first command in the Dockerfile
- RUN : used to execute a command during the build process of the image
- ADD : copy a file from host machine into a new docker image (or a URL)
- ENV : define an environment variable (e.g. JAVA_HOME)
- USER : specify the user which will be used to run any subsequent RUN instructions

  . . .

Building Our Virtual Cluster

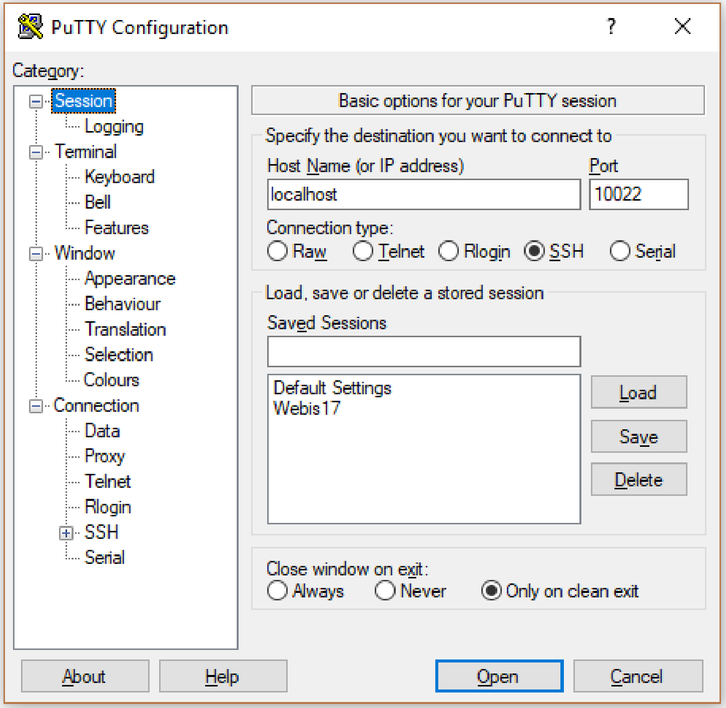
Step 1: SSH Gateway



Building Our Virtual Cluster

Step 1: SSH Gateway

- ❑ Review `Dockerfile.gateway`
- ❑ Run in terminal: `docker-compose up --build`
- ❑ Connect to the SSH Gateway `ssh -p 10022 tutorial@localhost`



PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): localhost

Port: 10022

Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

Load, save or delete a stored session

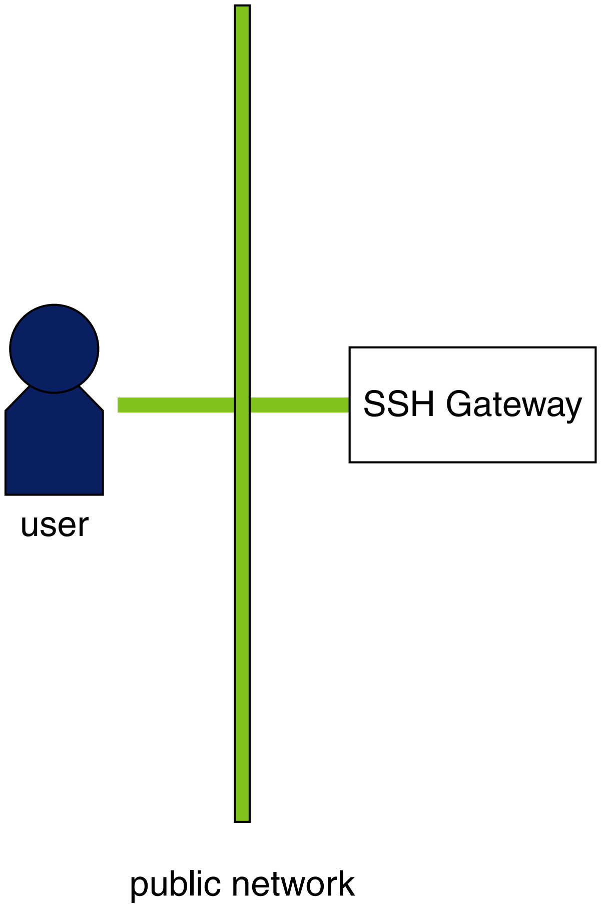
Saved Sessions:

Default Settings
Webis17

Close window on exit: ○ Always ○ Never ● Only on clean exit

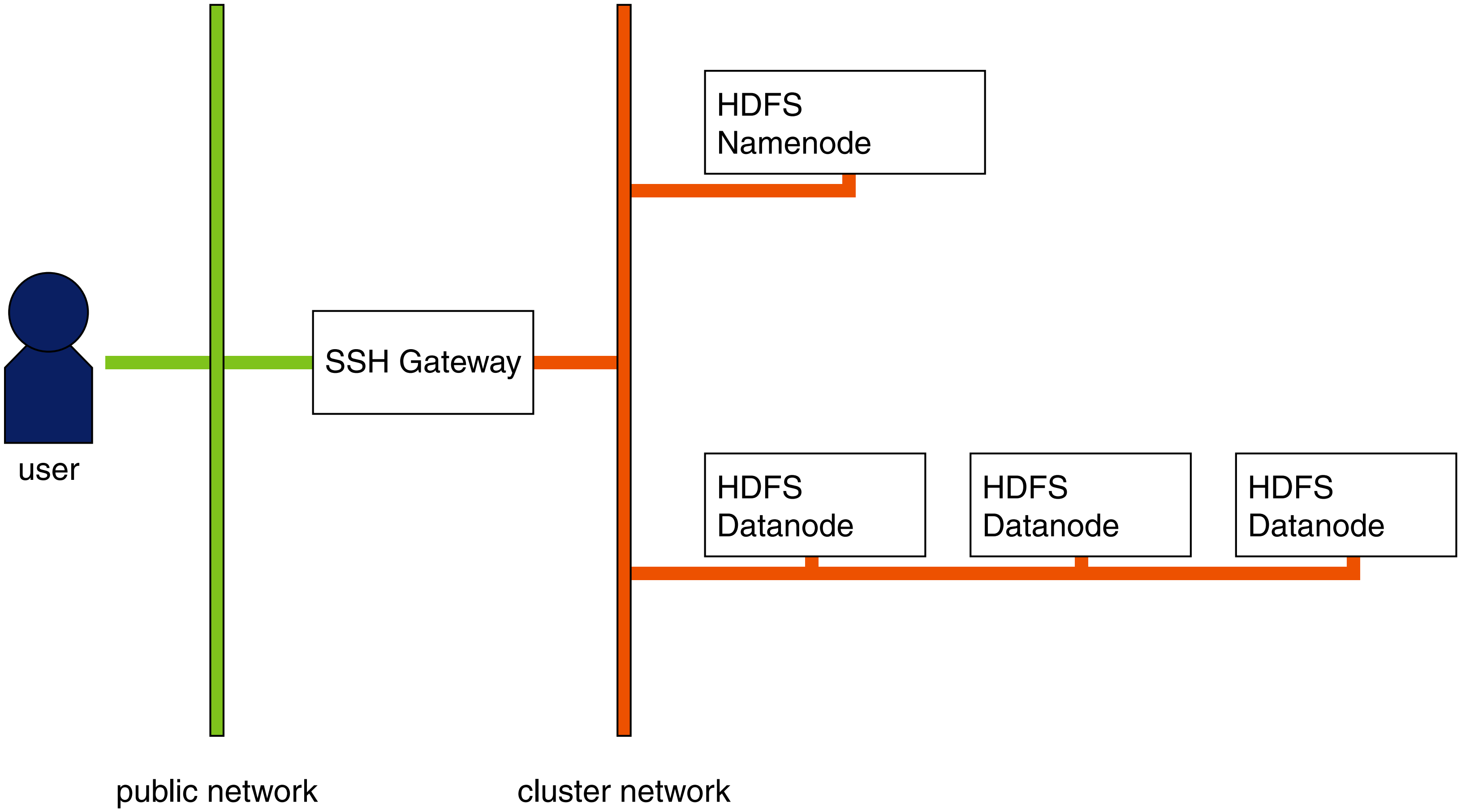
Building Our Virtual Cluster

Step 1: SSH Gateway



Building Our Virtual Cluster

Step 2: HDFS Distributed File System



Building Our Virtual Cluster

Step 2: HDFS Distributed File System

- Add namenode to virtual cluster
- Format the namenode:
  1. `docker-compose run --rm namenode bash`
  2. `hdfs namenode -format`
- Add datanodes to virtual cluster
- Re-start the virtual cluster:
  1. `docker-compose down`
  2. `docker-compose up`
- Set up proxy access to HDFS web UI
- Review configuration files
- Basic HDFS commands + web UI

  `http://namenode:50070`

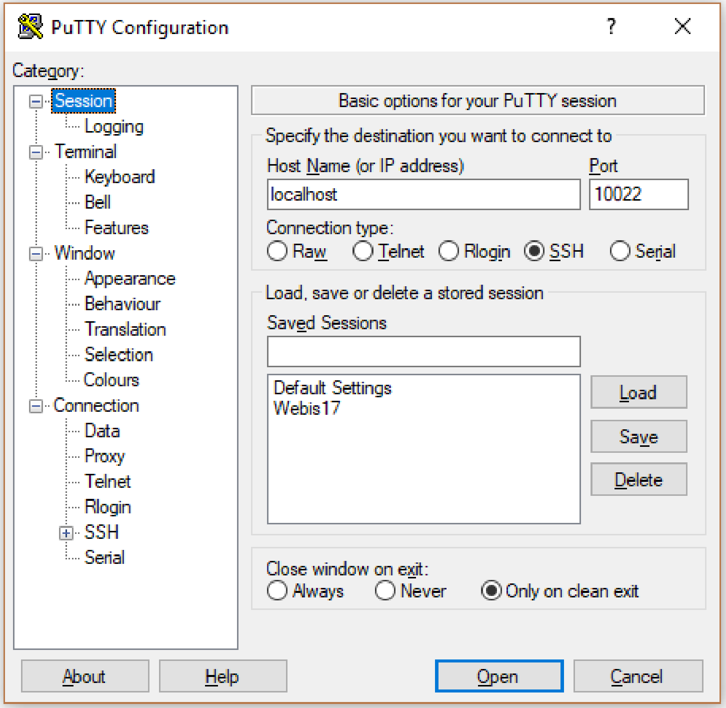
Building Our Virtual Cluster

SSH configurations

For ssh-client in **Linux & macOs** :

```
ssh -p 10022 -D 12345 tutorial@localhost
```

For **Windows** using Putty:



Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): localhost

Port: 10022

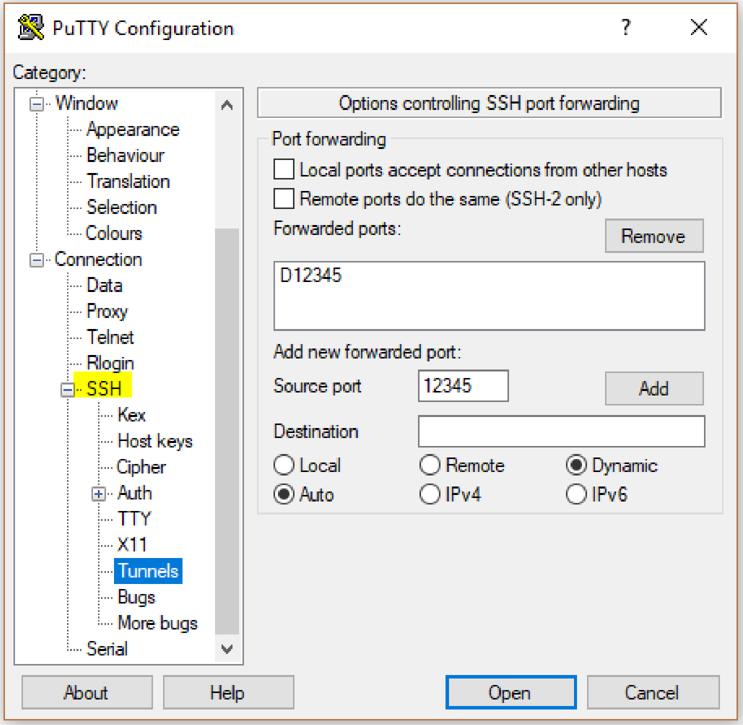
Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

Load, save or delete a stored session

Saved Sessions

Default Settings
Webis17

Close window on exit: ○ Always ○ Never ● Only on clean exit



Options controlling SSH port forwarding

Port forwarding

□ Local ports accept connections from other hosts

□ Remote ports do the same (SSH-2 only)

Forwarded ports:

D12345

Add new forwarded port:

Source port: 12345

Destination:

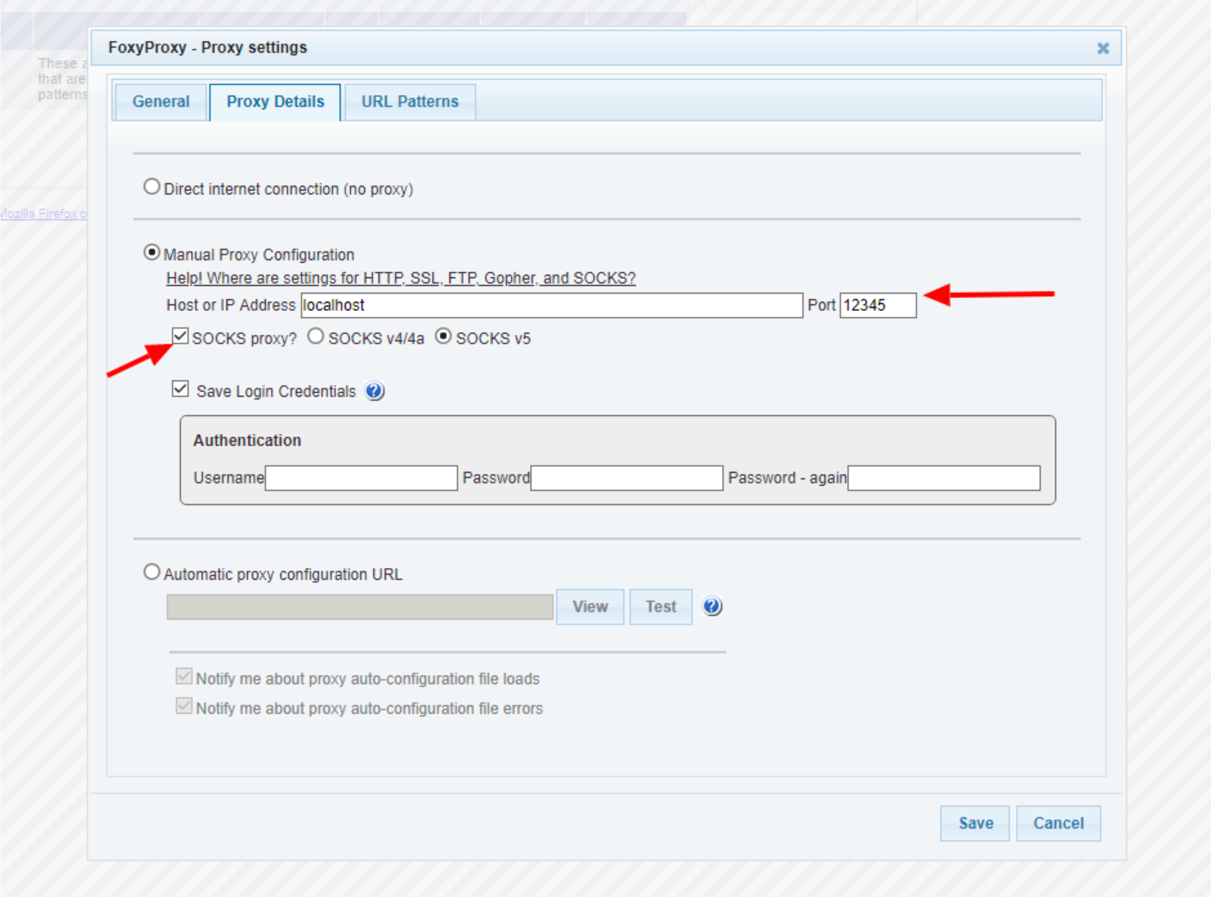
○ Local ○ Remote ● Dynamic

● Auto ○ IPv4 ○ IPv6

Building Our Virtual Cluster

SSH configurations

FoxyProxy configuration in the browser



FoxyProxy - Proxy settings

○ Direct internet connection (no proxy)

● Manual Proxy Configuration

Help! Where are settings for HTTP, SSL, FTP, Gopher, and SOCKS?

Host or IP Address localhost Port 12345

☑ SOCKS proxy? ○ SOCKS v4/4a ● SOCKS v5

☑ Save Login Credentials

Authentication

Username Password Password - again

○ Automatic proxy configuration URL

☑ Notify me about proxy auto-configuration file loads

☑ Notify me about proxy auto-configuration file errors

Building Our Virtual Cluster

SSH configurations

If you're running "Docker Toolbox" (e.g. on Windows versions earlier than 10), you need an additional step:

- ❑ Open "Oracle VirtualBox"
- ❑ Select "Settings" for the Docker virtual machine
- ❑ Select "Network" → "Advanced" → "Port Forwarding"
- ❑ Create a new port forwarding rule (top right button)
- ❑ For the new rule, change both "Host port" and "Guest port" to 10022 (leave the other fields as they are)

Building Our Virtual Cluster

Basic HDFS Commands

When logged into the *gateway* node, you can now run the following commands:

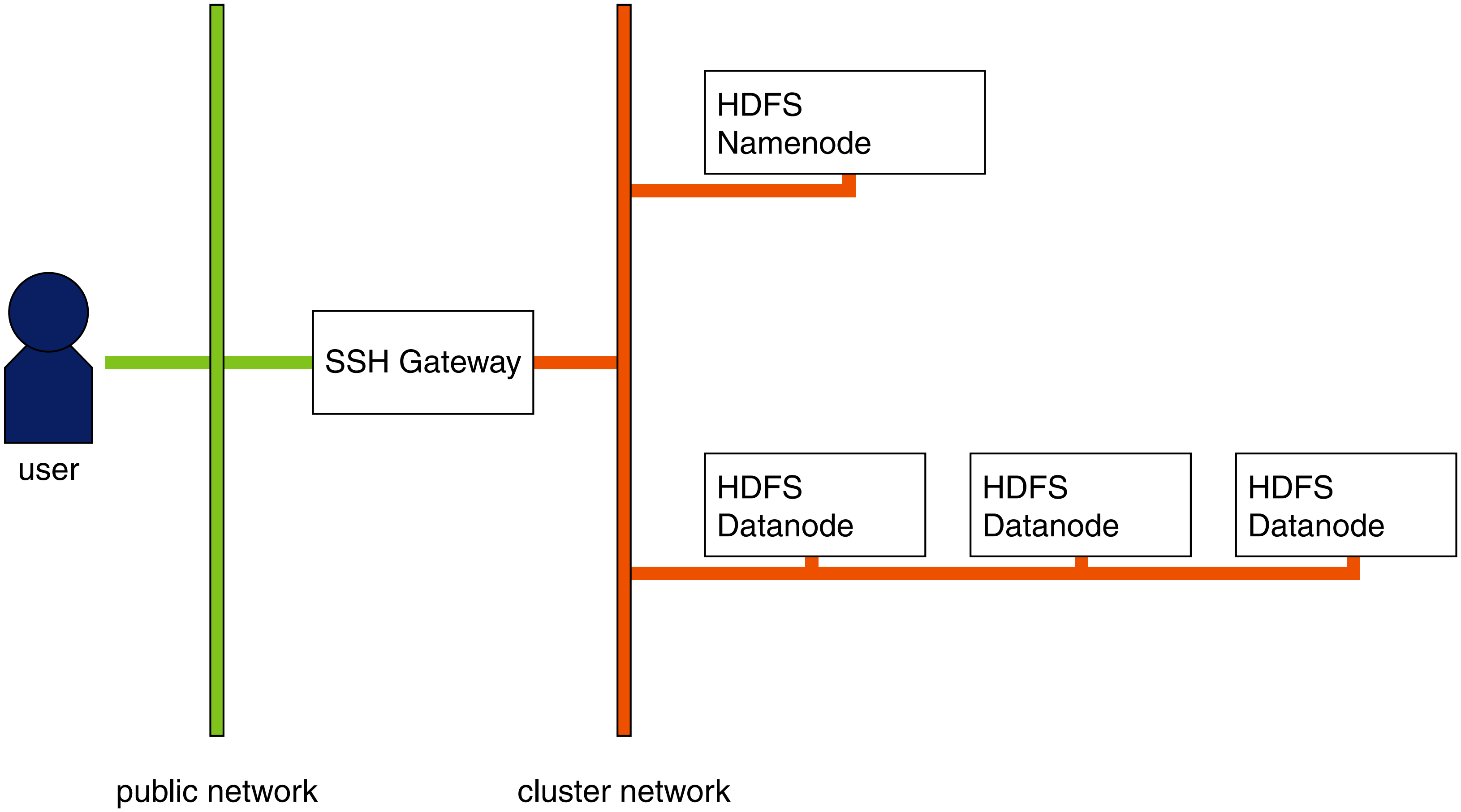
List files	`hadoop fs -ls NAME`
Remove directory	`hadoop fs -rmdir NAME`
Remove file	`hadoop fs -rm NAME`
Copy from local FS to HDFS	`hadoop fs -put LOCAL REMOTE`

Create a HDFS home directory for your user for later:

```
hadoop fs -mkdir -p /user/tutorial
```

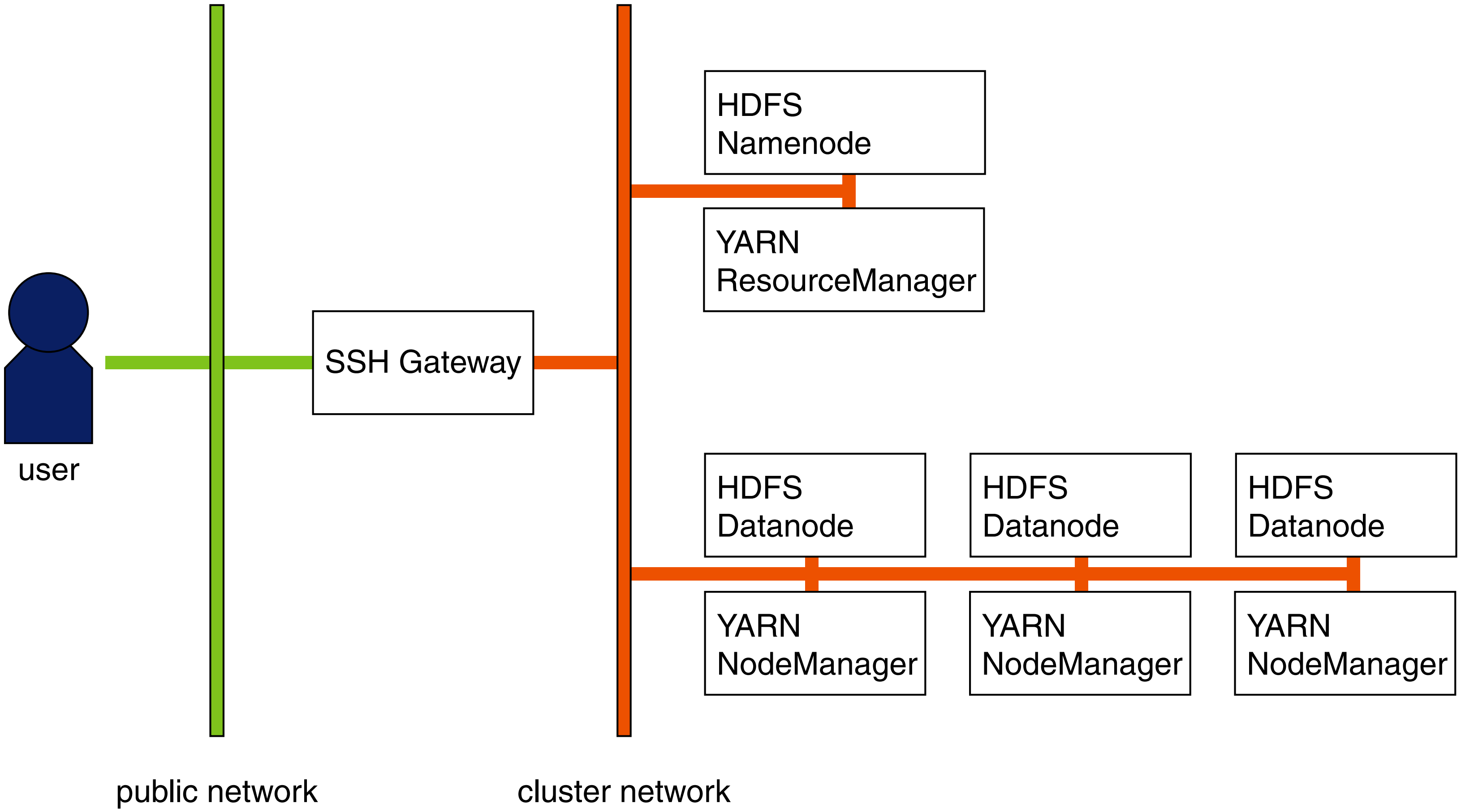
Building Our Virtual Cluster

Step 2: HDFS Distributed File System



Building Our Virtual Cluster

Step 3: YARN Distributed Processing Framework



Building Our Virtual Cluster

Step 3: YARN Distributed Processing Framework

- ❑ Add YARN processes to virtual cluster
- ❑ Review configuration files
- ❑ Explore ResourceManager Web UI:
  `http://resourcemanager:8088`
- ❑ Continue with MapReduce...

MapReduce

Problem

- Collecting data is easy and cheap
- Evaluating data is difficult

Solution

- Divide and Conquer
- Parallel Processing

MapReduce

MapReduce Steps

1. **Map** Each worker applies the `map()` function to the local data and writes the output to temporary storage. Each output record gets a key.

2. **Shuffle** Worker nodes redistribute data based on the output keys: all records with the same key go to the same worker node.

3. **Reduce** Workers apply the `reduce()` function to each group, per key, in parallel.

The user specifies the `map()` and `reduce()` functions

MapReduce

Example: Counting Words

Mary had a
little lamb

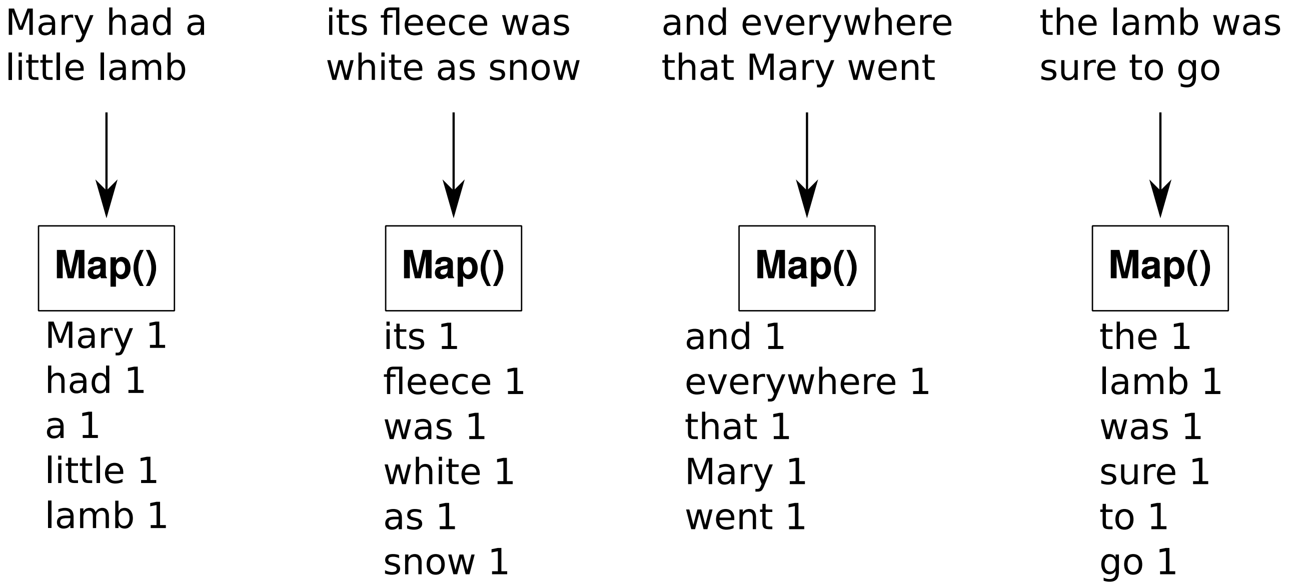
its fleece was
white as snow

and everywhere
that Mary went

the lamb was
sure to go

MapReduce

Example: Counting Words



Mary had a little lamb

Map()

Mary 1
had 1
a 1
little 1
lamb 1

its fleece was white as snow

Map()

its 1
fleece 1
was 1
white 1
as 1
snow 1

and everywhere that Mary went

Map()

and 1
everywhere 1
that 1
Mary 1
went 1

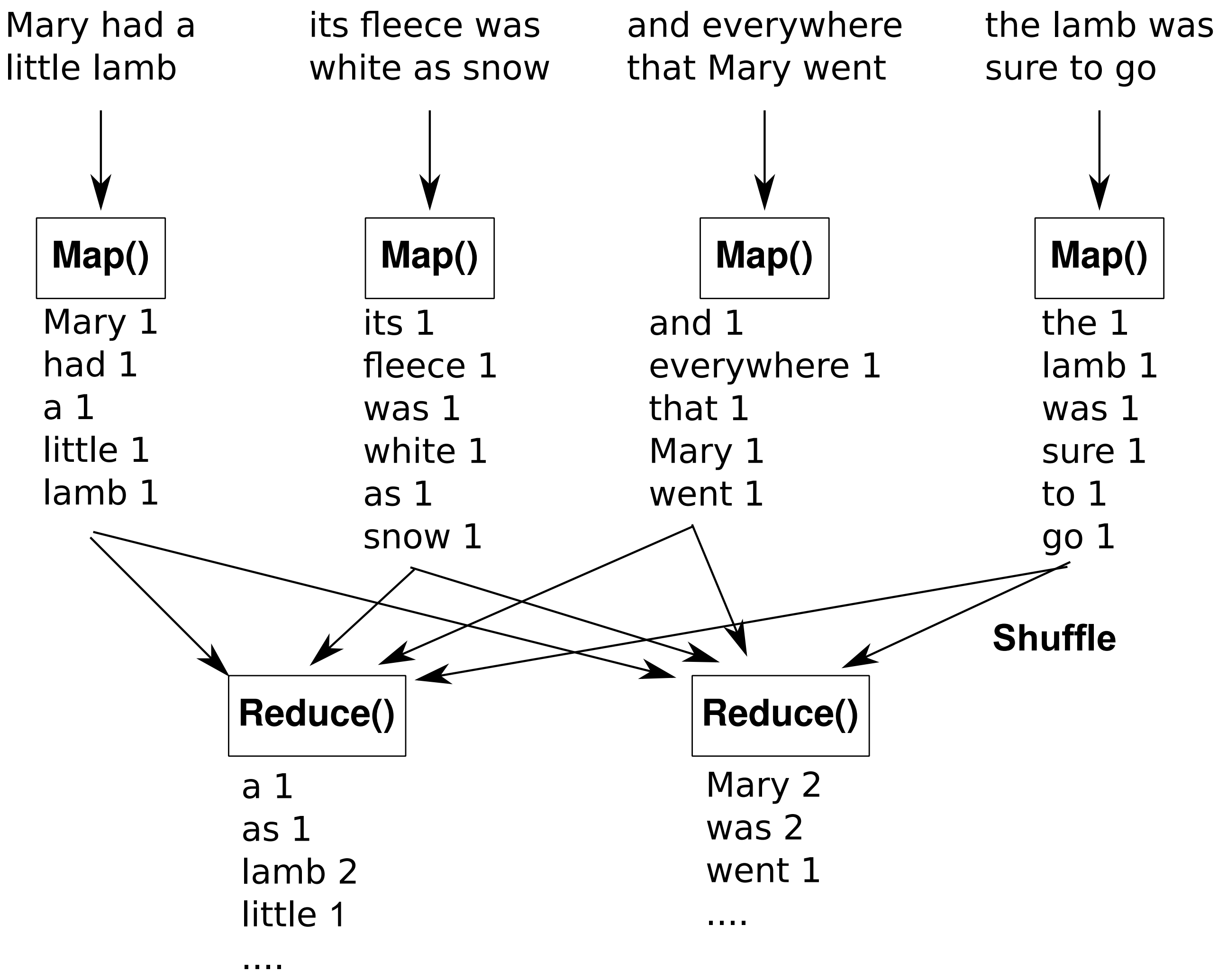
the lamb was sure to go

Map()

the 1
lamb 1
was 1
sure 1
to 1
go 1

MapReduce

Example: Counting Words



Mary had a
little lamb

Map()

Mary 1
had 1
a 1
little 1
lamb 1

its fleece was
white as snow

Map()

its 1
fleece 1
was 1
white 1
as 1
snow 1

and everywhere
that Mary went

Map()

and 1
everywhere 1
that 1
Mary 1
went 1

the lamb was
sure to go

Map()

the 1
lamb 1
was 1
sure 1
to 1
go 1

Shuffle

Reduce()

a 1
as 1
lamb 2
little 1
....

Reduce()

Mary 2
was 2
went 1
....

MapReduce

Data Representation with Key-Value Pairs

Map Step:

`Map(k1,v1)` → `list(k2,v2)`

Sorting and Shuffling:

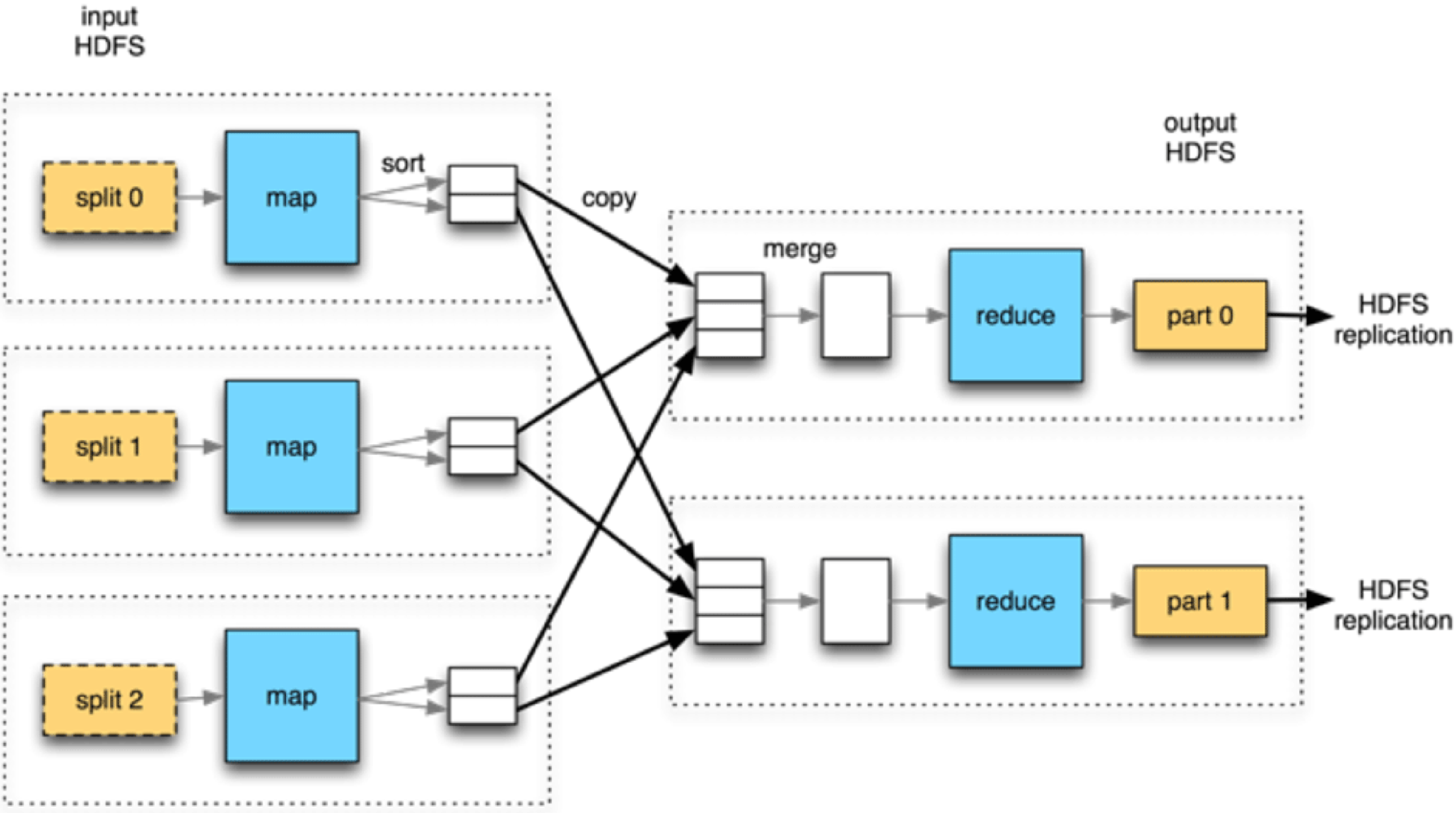
All pairs with the same key are grouped together; one group per key.

Reduce Step:

`Reduce(k2, list(v2))` → `list(v3)`

MapReduce

MapReduce on YARN



MapReduce

MapReduce on YARN

Recap: Components of the YARN Framework

- **ResourceManager** Single instance per cluster, controls container allocation
- **NodeManager** Runs on each cluster node, provides containers to applications

Components of a YARN MapReduce Job

- **ApplicationMaster** Controls execution on the cluster (one for each YARN application)
- **Mapper** Processes input data
- **Reducer** Processes (sorted) Mapper output

Each of the above runs in a YARN Container

MapReduce

MapReduce on YARN

Basic process:

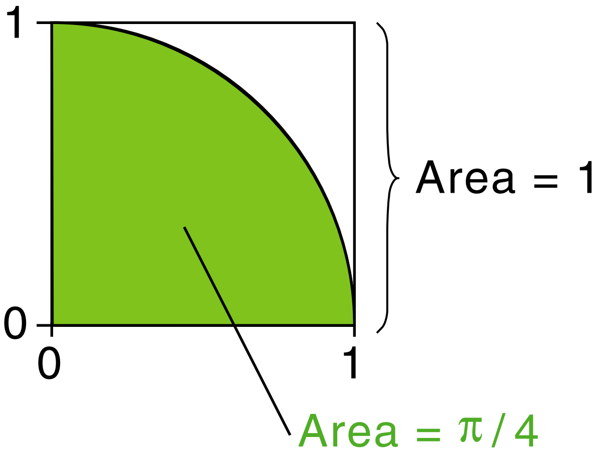
1. Client application requests a container for the ApplicationMaster
2. ApplicationMaster runs on the cluster, requests further containers for Mappers and Reducers
3. Mappers execute user-provided `map()` function on their part of the input data
4. The `shuffle()` phase is started to distribute map output to reducers
5. Reducers execute user-provided `reduce()` function on their group of map output
6. Final result is stored in HDFS

See also: [Anatomy of a MapReduce Job]

MapReduce Examples

Quasi-Monte-Carlo Estimation of π

Idea:

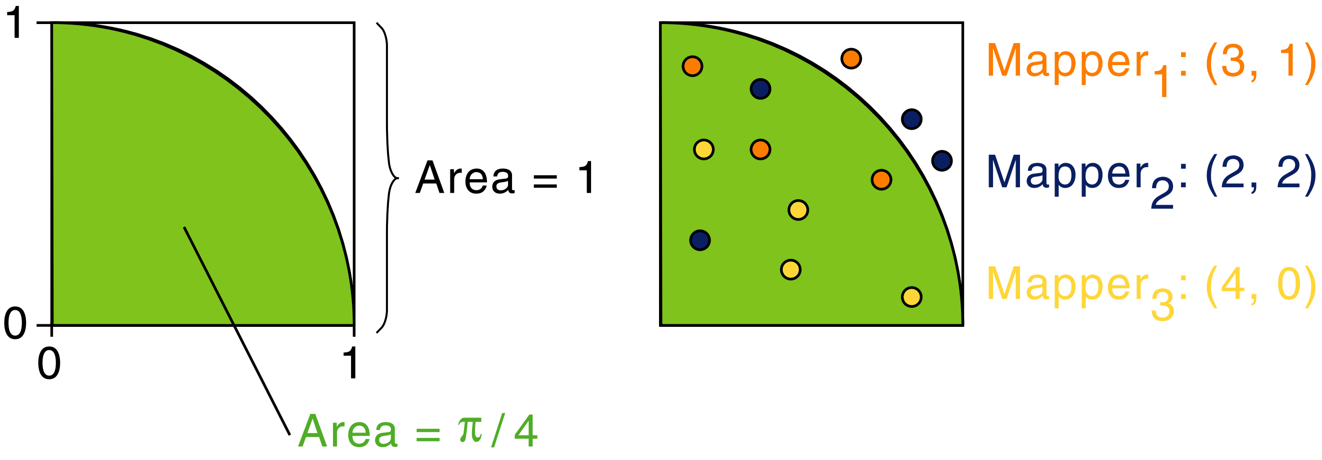


- The area of a circle segment inside the unit square is $\frac{\pi}{4}$

MapReduce Examples

Quasi-Monte-Carlo Estimation of π

Idea:

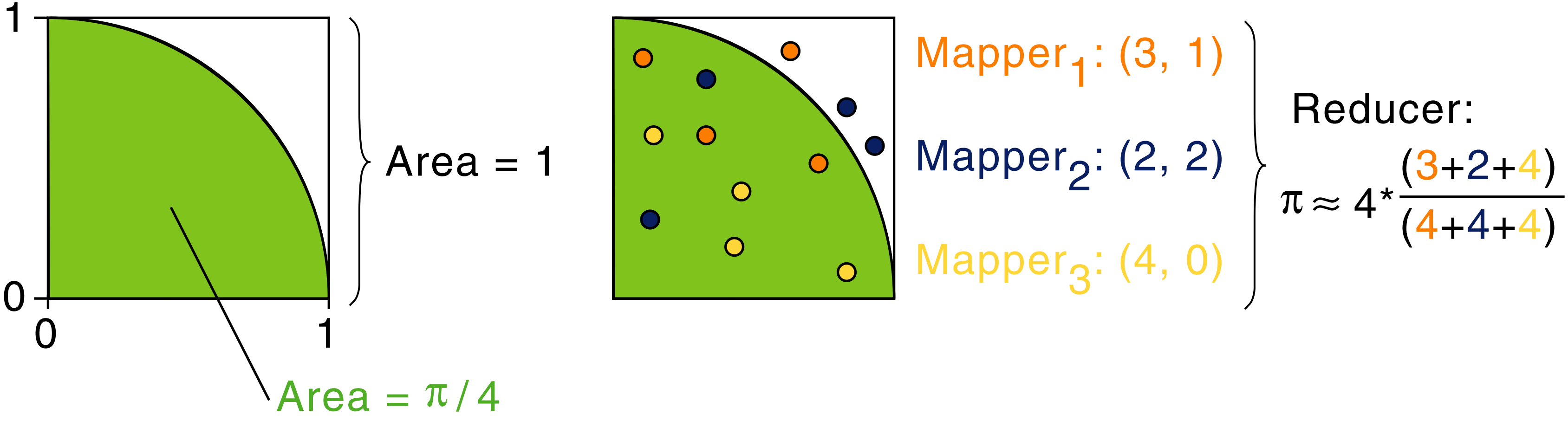


- The area of a circle segment inside the unit square is $\frac{\pi}{4}$
- Each mapper generates some random points inside the square, and counts how many fall inside/outside the circle segment.

MapReduce Examples

Quasi-Monte-Carlo Estimation of π

Idea:



Mapper$_1$: (3, 1)

Mapper$_2$: (2, 2)

Mapper$_3$: (4, 0)

Reducer:

$$\pi \approx 4 * \frac{(3+2+4)}{(4+4+4)}$$

- The area of a circle segment inside the unit square is $\frac{\pi}{4}$
- Each mapper generates some random points inside the square, and counts how many fall inside/outside the circle segment.
- The reducer sums up points inside and points total, to compute our estimate of π.

MapReduce Examples

Monte-Carlo Estimation of π

This is already included as an example program in Hadoop!

Connect to the *gateway* node and run:

```
cd /opt/hadoop-*/share/hadoop/mapreduce
```

then:

```
hadoop jar hadoop-mapreduce-examples-*.jar pi 4 1000000
```

The output should look like this:

```
Number of Maps = 4
Samples per Map = 1000000
Wrote input for Map #0
...
Job Finished in 13.74 seconds
Estimated value of Pi is 3.14160400000000000000
```

MapReduce Examples

Parallellizing Shell Scripts with Hadop Streaming

Let’s say we want to know which of the words “you” and “thou” occurs more frequently in Shakespeare’s works.

We can answer our question with simple Linux shell script. First some basics.

Download the file `shakespeare.txt` to the `workspace` directory of your *gateway* node.

Then, connect to the *gateway* node.

MapReduce Examples

Some Shell Scripting Basics

`cat FILE` — outputs contents of FILE

`A | B` — the output of command A becomes input of command B

`grep PATTERN` — outputs all input lines containing PATTERN

MapReduce Examples

Some Shell Scripting Basics

`cat FILE` — outputs contents of FILE

`A | B` — the output of command A becomes input of command B

`grep PATTERN` — outputs all input lines containing PATTERN

Example:

```
cat shakespeare.txt | grep ’ you ’
```

MapReduce Examples

Some Shell Scripting Basics

`cat FILE` — outputs contents of FILE
`A | B` — the output of command A becomes input of command B
`grep PATTERN` — outputs all input lines containing PATTERN

Example:

```
cat shakespeare.txt | grep ' you '
```

`grep -o PATTERN` — outputs only the matching part of each input line.
`\|` — inside the PATTERN, marks an alternative ("or")

MapReduce Examples

Some Shell Scripting Basics

`cat FILE` — outputs contents of FILE

`A | B` — the output of command A becomes input of command B

`grep PATTERN` — outputs all input lines containing PATTERN

Example:

```
cat shakespeare.txt | grep ’ you ’
```

`grep -o PATTERN` — outputs only the matching part of each input line.

`\|` — inside the PATTERN, marks an alternative (“or”)

Example:

```
cat shakespeare.txt | grep -o ’ you \| thou ’
```

MapReduce Examples

Some Shell Scripting Basics

`cat FILE` — outputs contents of FILE

`A | B` — the output of command A becomes input of command B

`grep PATTERN` — outputs all input lines containing PATTERN

Example:

```
cat shakespeare.txt | grep ’ you ’
```

`grep -o PATTERN` — outputs only the matching part of each input line.

`\|` — inside the PATTERN, marks an alternative (“or”)

Example:

```
cat shakespeare.txt | grep -o ’ you \| thou ’
```

`sort` — sorts the input alphabetically

`uniq -c` — counts consecutive identical lines in the input

MapReduce Examples

Some Shell Scripting Basics

`cat FILE` — outputs contents of FILE

`A | B` — the output of command A becomes input of command B

`grep PATTERN` — outputs all input lines containing PATTERN

Example:

```
cat shakespeare.txt | grep ’ you ’
```

`grep -o PATTERN` — outputs only the matching part of each input line.

`\|` — inside the PATTERN, marks an alternative (“or”)

Example:

```
cat shakespeare.txt | grep -o ’ you \| thou ’
```

`sort` — sorts the input alphabetically

`uniq -c` — counts consecutive identical lines in the input

So, finally:

[full explanation]

```
cat shakespeare.txt | grep -o ’ you \| thou ’ | sort | uniq -c
```

MapReduce Examples

Parallellizing Shell Scripts with Hadop Streaming

We have our answer, but we only used one machine. Hadoop Streaming lets us easily parallelize such shell scripts over the entire cluster!

MapReduce Examples

Parallellizing Shell Scripts with Hadop Streaming

We have our answer, but we only used one machine. Hadoop Streaming lets us easily parallelize such shell scripts over the entire cluster!

1. Put the input file in HDFS:

   ```
   hadoop fs -put shakespeare.txt shakespeare-hdfs.txt
   ```

2. Go to the directory with the Hadoop Streaming Jar file:

   ```
   cd /opt/hadoop-*/share/hadoop/tools/lib
   ```

3. Run our shellscript as a Streaming job:

   ```
   hadoop jar hadoop-streaming-*.jar \
       -input shakespeare-hdfs.txt \
       -output my-word-counts \
       -mapper "grep -o ’ you \| thou ’" \
       -reducer "uniq -c"
   ```

Notes: \ means “continue the previous line”; Hadoop already does the sorting for us.

MapReduce Examples

Parallellizing Shell Scripts with Hadop Streaming

Let’s look at the results:

```
hadoop fs -ls my-word-counts
```

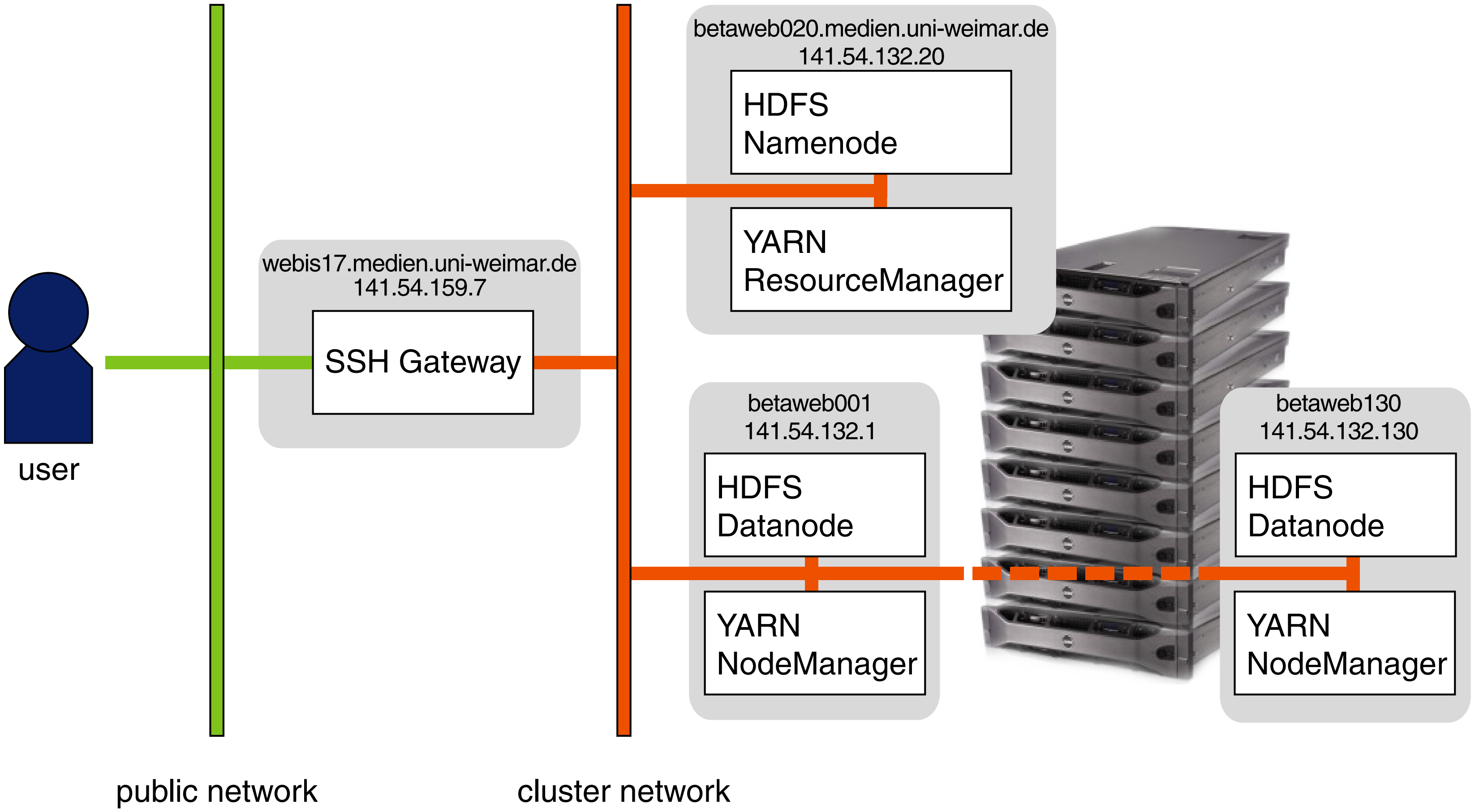
```
Found 2 items
-rw-r-r- 3 tutorial tutorial 0 2018-04-21 13:41 my-word-counts/_SUCCESS

-rw-r-r- 3 tutorial tutorial 31 2018-04-21 13:41 my-word-counts/part-00000
```

```
hadoop fs -cat my-word-counts/part-00000
```

```
4159 thou
8704 you
```

Working with the Real Cluster

Betaweb Cluster Network Layout



Working with the Real Cluster

Things to Know

Gateway host	`webis17.medien.uni-weimar.de`
Gateway login	(your university username)
Gateway password	(check your email)

ResourceManager UI	`http://betaweb020.medien.uni-weimar.de:8088`
HDFS UI	`http://betaweb020.medien.uni-weimar.de:50070`

Python Notebook UI	`https://webis17.medien.uni-weimar.de:8000`