Hands-on session: set up a big data architecture

Fabio Viola

For your convenience:

`https://baltig.infn.it/fviola/corsoccr---hands-on-big-data`

Aim of this session

Create an infrastructure to process local log les with Apache's Big Data tools.

More in particular:

- Flume will read data from syslog;
- Kafka will dispatch Flume's output;
- Spark will be used to perform basic analysis; tasks;
- In uxDB to write results of real-time analyses (hands-o);
- Grafana will be used to plot data (hands-o).

Pre-requisites

In this session I'll often refer to tmux. This tool is essential for mainly two reasons:

- allows switching among multiple remote terminals keeping just an ssh session opened.
- keeping alive (and re-attach to) running processes even after shutting down an ssh session (. . . also useful in case of network failures).

Let's install tmux with:

```
$ sudo yum install tmux
```

So, right after your first ssh connection open tmux with:

```
$ tmux
```

The next slide will provide an insight on the main shortcuts.

Tmux essentials

tmux provides a wide set of commands. We will focus on a few of them, just to perform the basic tasks. Inside a tmux session, remember these three shortcuts:

Ctrl-b s	List of active tmux sessions
Ctrl-b :new	Create a new tmux session
Ctrl-b $	Rename the current session

Outside a tmux session, to open the tool and connect to an existing session we just need:

```
$ tmux attach
```

This will attach to the last active session, and we would then be able to switch using Ctrl-b s .

Tmux in short...

Using tmux is like making Alt-tab among your windows... but among your terminal sessions!

Source: https://i.stack.imgur.com

Text editor

You will absolutely need a text editor. In this presentation I will refer to GNU Emacs, but other choices are available on your system (i.e. vi...) or can be installed with yum.

To get started with Emacs, just remember these easy shortcuts:

Ctrl-x Ctrl-f	to open a file
Ctrl-x Ctrl-s	to save a file
Ctrl-x Ctrl-c	to quit
Ctrl-_	to undo a change
Shift-INS	to paste from the clipboard

Disclaimer: vi users will not be harmed during this tutorial.

Apache Flume

What is it?

Apache Flume is a data ingestion tool designed to e ciently handle large amounts of data coming fromheterogeneous sources and owing to heterogeneous recipients.

More in particular, Flume is able to retrieve (and write) data from (to): Avro, Thrift, Exec, JMS, Spool directories, Taildir, Twitter, Kafka, Netcat, and many other.... . .

Flume in short...

Source: http://kdam.iltrovatore.it/.jpg

Flume's model

Flume's model is based on three main components, as shown below:

Basically, every Flume application consists of a con guration le containing settings for the sources, channels and sinks.

Installation

1. Create a new tmux session for ume
2. Download

```
$ wget www.apache.org/.../apache-flume-VERSION-bin.tar.gz
```

3. Uncompress

```
$ tar xvzf apache-flume-VERSION-bin.tar.gz
```

Click here for current version.

Running Flume { 1

Before running Flume, we of course need to create a con guration le for it. For sure we will need at least a source, a channel and a sink:

```
a1.sources = r1
a1.channels = c1
a1.sinks = s1
```

a1 is an identi er for referring to our application.

Running Flume { 2

Let's suppose that we want to read log data produced by our GNU/Linux machine and simply output it on screen. . .

The source is a given log les (e.g/var/log/syslog). In a bash shell we could simply usetail -F . . . So we can selectexec as a source type and specify this command:

```
a1.sources.r1.type = exec
a1.sources.r1.command = tail -F /var/log/messages
a1.sources.r1.channels = c1
```

Running Flume { 3

Now we con gure our channel:

```
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100
```

Then the sink:

```
a1.sinks.s1.type = logger
a1.sinks.s1.channel = c1
```

Save this le with a custom name (e.g.flume1.conf).

Running Flume { 4

We are ready to start ume:

```
$ bin/flume-ng agent --conf conf --conf-file flume1.conf \
  --name a1 -Dflume.root.logger=INFO,console
```

Running Flume { 5

If the amount of generated log is too little, you can try with a log generator:

```
$ git clone https://baltig.infn.it/rossitisbeni/LogSimulator.git
```

And clone a repository with a set of log files:

```
$ git clone https://github.com/logpai/loghub
```

Then, from the log simulator folder, execute the code:

```
$ python logsimulator.py
```

Running Flume { 6

Our rst example is complete! Simple, isn't it? Now you can play with Flume using the sources and sinks you may need.

Supported sources are:

Avro, Thrift, Exec, JMS, Spool directories, Taildir, Twitter, Kafka, Netcat ...

Supported sinks are:

HDFS, Hive, Logger, Avro, Thrift, IRC, FileRoll, Null, HBaseSink, ElasticSearch ...

Resources

Homepage: https://flume.apache.org/

Flume user guide: https://flume.apache.org/releases/content/1.9.0/FlumeUserGuide.html

Introduction

Kafka is a topic-based publish-subscribe engine that is used as a broadcaster for the information in our system.

Rather than simply reading data from Flume, we prefer using Kafka in order to permit multiple clients at once.

Kafka in short...

Source: https://y.yarn.co

Installation

Create a new tmux session for kafka

Download and uncompress Kafka

```
$ wget http://www.apache.org/.../kafka_VERSION.tgz
$ tar xzvf kafka_VERSION.tgz
$ ln -s kafka_VERSION kafka
```

Click here for current version.

Running Kafka

Kafka requires zookeeper to be running, since it keeps track of status of the Kafka cluster nodes and it also keeps track of Kafka topics, partitions etcSo let's start zookeeper rst and then focus on Kafka:[1]

Create a new tmux session for zookeeper

Start zookeeper

```
$ cd kafka/bin
$ sh zookeeper-server-start.sh ../config/zookeeper.prop..
```

Switch back to kafka session and start kafka

```
$ cd kafka/bin
$ sh kafka-server-start.sh ../config/server.properties
```

Then, Zookeeper and Kafka will respectively be listening on ports 2181 and 9092.

[1] www.cloudkarafka.com/blog/2018-07-04-cloudkarafka_what_is_zookeeper.html

Creating a topic

To start publishing and subscribing to kafka, we first need to create one or more topics:

Start a new tmux session

Create a new topic

```
$ cd kafka/bin
$ sh kafka-topics.sh --create --bootstrap-server \
> localhost:9092 --replication-factor 1 --partitions 1 \
> --topic MyBigData
```

Now, we are ready to start publishing and subscribing. As a first step, we'll rely on the utilities provided with the Kafka package.

Subscribing to a topic

A subscription to the topic MyBigData can be performed with a very simple tool available in the bin folder of Apache's package:

Open a new tmux session (e.g. named "kafka-consumer")

Subscribe to the topic

```
$ cd kafka/bin
$ kafka-console-consumer.sh --bootstrap-server \
> localhost:9092 --topic MyBigData --from-beginning
```

Publishing on a given topic

Now let's publish some messages with the topic MyBigData using a very simple tool available in the bin folder of Apache's package:

Open a new tmux session (e.g. named "kafka-producer")

Publish a simple hello world message

```
$ cd kafka/bin
$ kafka-console-producer.sh --broker-list \
> localhost:9092 --topic MyBigData
Hello World
```

Now switch back to session "kafka-consumer" and see the magic...

Editing Flume con guration

Until now, we have been working with Kafka by manually producing its input.
It's time to connect Flume to Kafka. . .
To do this, open Flume con guration le and edit the sink to look like this:

```
a1.sinks.s1.type = kafka
a1.sinks.s1.channel = c1
a1.sinks.s1.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.s1.kafka.topic = MyBigData
a1.sinks.s1.kafka.bootstrap.servers = localhost:9092
a1.sinks.s1.kafka.flumeBatchSize = 20
a1.sinks.s1.kafka.producer.acks = 1
a1.sinks.s1.kafka.producer.linger.ms = 1
a1.sinks.s1.kafka.producer.compression.type = snappy
```

Then, restart Flume, and switch to your kafka consumer to see log lines dispatched by kafka.

Retention

Apache Kafka provides two types of Retention Policies:

Time-based retention : defaults to 7 days. When the time limit is hit, the segment is marked for deletion or compaction depending on con gured cleanup policy.

Size-based retention: obviously we set the maximum size for a segment. This policy is not popular as this does not provide good visibility about message expiry, but it is useful in a scenarios with a limited disk space.

Retention

Retention can be set in the con guration le with one of the following lines (reported in decreasing order of priority):

```
log.retention.ms=1680000
log.retention.minutes=1680
log.retention.hours=168
```

or selectively by altering a topic:

```
$ bin/kafka-topics.sh --zookeeper localhost:2181 \
> --alter --topic MyBigData --config retention.ms=1000
```

(where we can specify the time in ms, minutes and hours).

Cleanup policies

In Kafka, messages are not immediately removed after they are consumed. Instead, the con guration of each topic determines how much space the topic is permitted and how it is managed. Cleanup con guration is per topic. Possible policies are:

Delete: the default policy that deletes segments exceeding the maximum time or size.

Compact: this is used to perform Compaction on a topic. Compaction removes records of each partition where there are more recent updates with the same primary key.

Delete and compact: combination of the two previous modes.

Retention

What is Spark?

Apache Spark is a distributed general-purpose cluster-computing framework, providing an interface for parallel programming.

Spark pivots on the concept of RDD (Resilient Distributed Dataset) to deal with data. RDDs are multisets of data item that can be processed in a very efficient way...

Among the many libraries provided by Spark, an interesting one is MLlib dedicated to machine learning.

Spark in short...

Installation

Create a new tmux session

Download and uncompress Spark

```
$ wget http://www.apache.org/.../spark-VERSION.tgz
$ tar zvxf spark-VERSION.tgz
$ ln -s spark-VERSION spark
```

Modify our /.profile to add the following lines

```
export PATH=$PATH:$JAVA_HOME/bin
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
export PATH=/home/<UTENTE>/spark/bin:$PATH
export SPARK_LOCAL_IP=127.0.0.1
```

Close this tmux session (Ctrl-d) and re-create it

Start the spark shell

```
$ spark-shell
```

Click here for current version.

Running Spark programs

The spark shell is great if you want to test some code on the y using Scala. We will refer to Scala because in latest versions of Spark, python is unfortunately not supported.

In the following slides we will create some simple Spark programs (e.g. named testX.scala) and run it with:

```
$ spark-shell \
> --jars spark-streaming-kafka-0-10-assembly_2.11.jar \
> -i testX.scala
```

For your convenience, you'll nd the required jar in the provided repository.

Our rst program { 1

Let's start by importing all the required stu .

```
// import requirements
import org.apache.spark.streaming.Seconds
import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.
    PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.
    Subscribe
import org.apache.spark.streaming.StreamingContext
```

Our rst program { 2

Then, we con gure the parameters related to kafka and the topic of interest.

```
// configure kafka
val kafkaParams = Map[String, Object](
  "bootstrap.servers" -> "localhost:9092",
  "key.deserializer" -> "org.apache.kafka.common.serialization.
       StringDeserializer",
  "value.deserializer" -> "org.apache.kafka.common.serialization.
       StringDeserializer",
  "group.id" -> "use_a_separate_group_id_for_each_stream",
  "auto.offset.reset" -> "latest",
  "enable.auto.commit" -> (false: java.lang.Boolean)
)

// create a streaming context
val streamingContext = new StreamingContext(sc, Seconds(1))

// configure the stream
val topics = Array("MyBigData")
val stream = KafkaUtils.createDirectStream[String, String](
  streamingContext,
  PreferConsistent,
  Subscribe[String, String](topics, kafkaParams)
)
```

Our rst program { 3

```
// now process each record
val records = stream.map(record => record.value)

// create a new DStream containing only the error messages
val errors = records.filter(status => status.contains("Error"))
errors.print()

// start the computation
streamingContext.start()
streamingContext.awaitTermination()
```

Our second program { 1

Now let's operate in batch mode. . . For simplicity we will now work directly in the spark-shell.

```
// open the log file
val lines = sc.textFile("/var/log/syslog")

// create two new RDDs with lines containing
// either the word "error" or the word "denied"
val err = lines.filter(l => l.contains("error"))
val den = lines.filter(l => l.contains("denied"))

// join the two RDDs
val mergedRDD = err ++ den

// calculate the number of words in this RDD
// using map and reduce
mergedRDD.flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_+_)
mergedRDD.count
```

Flat VS Flatmap

Where to write its output?

Di erent needs, di erent options. . .

- HDFS
- Kafka
- In uxDB
- . . .

For the rest of the presentation we'll assume In uxDB as our default output.

What is In uxDB?

In uxDB is a time series database designed to handle high write and query loads. In uxDB has a line protocol for sending time series data which takes the following form:

```
measurement-name tag-set field-set timestamp
```

Conceptually you can think of a measurement as an SQL table, where the primary index is always time. tags and elds are e ectively columns in the table. tags are indexed, and elds are not.

In uxDB in short. . .

Source: https://i.eurosport.com/

Installation

For simplicity, you can rely on your package manager and type:

```
$ yum install influxdb
```

Execution { 1

Now, suppose we want to push our Spark real-time results in a proper In uxDB database. First of all, we need to create it:

```
$ influx
> CREATE DATABASE spark_res
```

Then, for a rapid test, we could check the existance of the DB with:

```
> SHOW DATABASES
```

and we could try storing a simple time series data with:

```
> USE spark_res
> INSERT errors,host=serverA,process=Storm value=76
```

Execution { 2

You can check the results of the previous command with a simple temporal query:

```
> USE spark_res
> SELECT "host","process","value" FROM "errors"
name: errors
------------
time                 host     process      value
1574082487851925338      serverA  storm    76
```

Now, we have all that's needed to feed our artist: Grafana!

What is Grafana?

Grafana is de ned as "the open observability platform". It is an easy tool to create dashboards plotting in real-time whatever you may desire.

Among its data sources, In uxDB is the most interesting one (for our purposes). Through temporal queries, Grafana constantly retrieves and plot data.

For time purposes, this part of the turorial will behands-o . . . :-P

Grafana in short. . .

Source: https://cff2.earth.com

Installation

Download and uncompress it

```
$ wget https://dl.grafana.com/oss/release/grafana-VERSION.tar.gz
$ tar -zxvf grafana-VERSION.tar.gz
$ cd grafana-VERSION
```

Con gure the server

```
$ cp conf/defaults.ini conf/custom.ini
$ emacs conf/custom.ini
```

Start the server

```
$ ./bin/grafana-server web
```

It will run by default on port 3000. By default an administrator user with login and password set tœadmin exists. After the rst login, the password must be changed.

Now?

Once the server is up and running, point your browser to http://localhost:3000 , then:

- Create a dashboard
- Create a new panel
- Set your temporal query to In ux
- Select the desired chart type

Example

Documentation

The following is a list of useful links to go deeper with the tools introduced in this presentation.

- Tmux documentation
- Apache Flume documentation
- Apache Kafka documentation
- RDD Documentation (Apache Spark)
- In uxDB documentation
- Grafana documentation