

simply easy learning

www.tutorialspoint.com



https://www.facebook.com/tutorialspointindia



https://twitter.com/tutorialspoint

About the Tutorial

Apache Flink is an open source stream processing framework, which has both batch and stream processing capabilities. Apache Flink is very similar to Apache Spark, but it follows stream-first approach. It is also a part of Big Data tools list. This tutorial explains the basics of Flink Architecture Ecosystem and its APIs.

Audience

This tutorial is for beginners who are aspiring to become experts with stream processing in Big Data Domain. It is also ideal for Big Data professionals who know Apache Hadoop and Apache Spark.

Prerequisites

Before proceeding with this tutorial, you should have basic knowledge of Scala programming and any Linux operating system.

Copyright & Disclaimer

© Copyright 2019 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com



Table of Contents

	About the Tutoriali
	Audiencei
	Prerequisitesi
	Copyright & Disclaimeri
	Table of Contentsii
1.	Apache Flink — Big Data Platform1
2.	Apache Flink — Batch vs Real-time Processing2
3.	Apache Flink — Introduction
	Ecosystem on Apache Flink
4.	Apache Flink — Architecture5
	Features of Apache Flink6
5.	Apache Flink — System Requirements7
6.	Apache Flink — Setup/Installation8
7.	Apache Flink — API Concepts
7.	Apache Flink — API Concepts
7.	
7. 8.	Dataset API10
	Dataset API
8. 9.	Dataset API
8. 9. 10.	Dataset API 10 DataStream API 11 Apache Flink — Table API and SQL 13 Apache Flink — Creating a Flink Application 14
8. 9. 10.	Dataset API 10 DataStream API 11 Apache Flink — Table API and SQL 13 Apache Flink — Creating a Flink Application 14 Apache Flink — Running a Flink Program 30
8. 9. 10.	Dataset API 10 DataStream API 11 Apache Flink — Table API and SQL 13 Apache Flink — Creating a Flink Application 14 Apache Flink — Running a Flink Program 30 Apache Flink — Libraries 32
8. 9. 10. 11.	Dataset API 10 DataStream API 11 Apache Flink — Table API and SQL 13 Apache Flink — Creating a Flink Application 14 Apache Flink — Running a Flink Program 30 Apache Flink — Libraries 32 Complex Event Processing (CEP) 32
8. 9. 10. 11.	Dataset API 10 DataStream API 11 Apache Flink — Table API and SQL 13 Apache Flink — Creating a Flink Application 14 Apache Flink — Running a Flink Program 30 Apache Flink — Libraries 32 Complex Event Processing (CEP) 32 Gelly 33
8. 9. 10. 11.	Dataset API 10 DataStream API 11 Apache Flink — Table API and SQL 13 Apache Flink — Creating a Flink Application 14 Apache Flink — Running a Flink Program 30 Apache Flink — Libraries 32 Complex Event Processing (CEP) 32 Gelly 33 Apache Flink — Machine Learning 35



14.	Apache Flink — Flink vs Spark vs Hadoop	39
15.	Apache Flink — Conclusion	40



1. Apache Flink — Big Data Platform

The advancement of data in the last 10 years has been enormous; this gave rise to a term 'Big Data'. There is no fixed size of data, which you can call as big data; any data that your traditional system (RDBMS) is not able to handle is Big Data. This Big Data can be in structured, semi-structured or un-structured format. Initially, there were three dimensions to data – Volume, Velocity, Variety. The dimensions have now gone beyond just the three Vs. We have now added other Vs – Veracity, Validity, Vulnerability, Value, Variability, etc.

Big Data led to the emergence of multiple tools and frameworks that help in the storage and processing of data. There are a few popular big data frameworks such as Hadoop, Spark, Hive, Pig, Storm and Zookeeper. It also gave opportunity to create Next Gen products in multiple domains like Healthcare, Finance, Retail, E-Commerce and more.

Whether it is an MNC or a start-up, everyone is leveraging Big Data to store and process it and take smarter decisions.



In terms of Big Data, there are two types of processing:

- Batch Processing
- Real-time Processing

Processing based on the data collected over time is called Batch Processing. For example, a bank manager wants to process past one-month data (collected over time) to know the number of cheques that got cancelled in the past 1 month.

Processing based on immediate data for instant result is called Real-time Processing. For example, a bank manager getting a fraud alert immediately after a fraud transaction (instant result) has occurred.

The table given below lists down the differences between Batch and Real-Time Processing:

Batch Processing	Real-Time Processing
Static Files	Event Streams
Processed Periodically in minute, hour, day etc.	Processed immediately (nanoseconds)
Past data on disk storage	In Memory Storage
Example: Bill Generation	Example: ATM Transaction Alert

These days, real-time processing is being used a lot in every organization. Use cases like fraud detection, real-time alerts in healthcare and network attack alert require real-time processing of instant data; a delay of even few milliseconds can have a huge impact.

An ideal tool for such real time use cases would be the one, which can input data as stream and not batch. Apache Flink is that real-time processing tool.



Apache Flink is a real-time processing framework which can process streaming data. It is an open source stream processing framework for high-performance, scalable, and accurate real-time applications. It has true streaming model and does not take input data as batch or micro-batches.

Apache Flink was founded by Data Artisans company and is now developed under Apache License by Apache Flink Community. This community has over 479 contributors and 15500 + commits so far.

Ecosystem on Apache Flink

The diagram given below shows the different layers of Apache Flink Ecosystem:

API's and LIBRARIES	Flink ML Gelly Table	Table	
API's an	Dataset AP Batch Process		aStream API Im Processing
KERNEL	Dis	Runtime tributed Streaming DataFlo	w
DEPLOY	Local Single JVM	Cluster Standalone, YARN, Mesos, Tez	Cloud Google GCE Amazon EC2
STORAGE	Files Local FS, HDFS, S3	Databases MongoDB, HBase, SQL 	Stream RabbitMQ, Kafka, Flume

Storage

Apache Flink has multiple options from where it can Read/Write data. Below is a basic storage list:

- HDFS (Hadoop Distributed File System)
- Local File System



- S3
- RDBMS (MySQL, Oracle, MS SQL etc.)
- MongoDB
- HBase
- Apache Kafka
- Apache Flume

Deploy

You can deploy Apache Fink in local mode, cluster mode or on cloud. Cluster mode can be standalone, YARN, MESOS.

On cloud, Flink can be deployed on AWS or GCP.

Kernel

This is the runtime layer, which provides distributed processing, fault tolerance, reliability, native iterative processing capability and more.

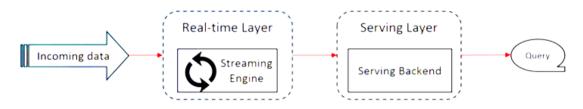
APIs & Libraries

This is the top layer and most important layer of Apache Flink. It has Dataset API, which takes care of batch processing, and Datastream API, which takes care of stream processing. There are other libraries like Flink ML (for machine learning), Gelly (for graph processing), Tables for SQL. This layer provides diverse capabilities to Apache Flink.



Apache Flink works on Kappa architecture. Kappa architecture has a single processor - stream, which treats all input as stream and the streaming engine processes the data in real-time. Batch data in kappa architecture is a special case of streaming.

The following diagram shows the Apache Flink Architecture.

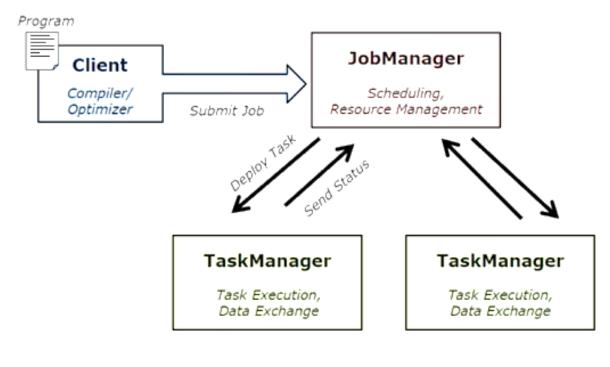


The key idea in Kappa architecture is to handle both batch and real-time data through a single stream processing engine.

Most big data framework works on Lambda architecture, which has separate processors for batch and streaming data. In Lambda architecture, you have separate codebases for batch and stream views. For querying and getting the result, the codebases need to be merged. Not maintaining separate codebases/views and merging them is a pain, but Kappa architecture solves this issue as it has only one view – real-time, hence merging of codebase is not required.

That does not mean Kappa architecture replaces Lambda architecture, it completely depends on the use-case and the application that decides which architecture would be preferable.

The following diagram shows Apache Flink job execution architecture.





Program

It is a piece of code, which you run on the Flink Cluster.

Client

It is responsible for taking code (program) and constructing job dataflow graph, then passing it to JobManager. It also retrieves the Job results.

JobManager

After receiving the Job Dataflow Graph from Client, it is responsible for creating the execution graph. It assigns the job to TaskManagers in the cluster and supervises the execution of the job.

TaskManager

It is responsible for executing all the tasks that have been assigned by JobManager. All the TaskManagers run the tasks in their separate slots in specified parallelism. It is responsible to send the status of the tasks to JobManager.

Features of Apache Flink

The features of Apache Flink are as follows:

- It has a streaming processor, which can run both batch and stream programs.
- It can process data at lightning fast speed.
- APIs available in Java, Scala and Python.
- Provides APIs for all the common operations, which is very easy for programmers to use.
- Processes data in low latency (nanoseconds) and high throughput.
- Its fault tolerant. If a node, application or a hardware fails, it does not affect the cluster.
- Can easily integrate with Apache Hadoop, Apache MapReduce, Apache Spark, HBase and other big data tools.
- In-memory management can be customized for better computation.
- It is highly scalable and can scale upto thousands of node in a cluster.
- Windowing is very flexible in Apache Flink.
- Provides Graph Processing, Machine Learning, Complex Event Processing libraries.



5. Apache Flink — System Requirements

The following are the system requirements to download and work on Apache Flink -

Recommended Operating System

- Microsoft Windows 10
- Ubuntu 16.04 LTS
- Apple macOS 10.13/High Sierra

Memory Requirement

- Memory Minimum 4 GB, Recommended 8 GB
- Storage Space 30 GB

Note – Java 8 must be available with environment variables already set.



6. Apache Flink — Setup/Installation

Before the start with the setup/ installation of Apache Flink, let us check whether we have Java 8 installed in our system.

Java - version

```
wubuntu@ubuntu-VirtualBox:~
ubuntu@ubuntu-VirtualBox:~$ java -version
java version "1.8.0_201"
Java(TM) SE Runtime Environment (build 1.8.0_201-b09)
Java HotSpot(TM) Client VM (build 25.201-b09, mixed mode)
ubuntu@ubuntu-VirtualBox:~$
```

We will now proceed by downloading Apache Flink.



ubuntu@ubuntu-VirtualBox:~\$ wget http://mirrors.estointernet.in/apache/flink/fli nk-1.7.1/flink-1.7.1-bin-scala_2.11.tgz --2019-01-24 23:11:34-- http://mirrors.estointernet.in/apache/flink/flink-1.7.1 /flink-1.7.1-bin-scala_2.11.tgz Resolving mirrors.estointernet.in (mirrors.estointernet.in)... 103.11.81.206, 24 04:9d00:0:1::ffff Connecting to mirrors.estointernet.in (mirrors.estointernet.in)|103.11.81.206|:8 0... connected. HTTP request sent, awaiting response... 200 OK Length: 258201561 (246M) [application/octet-stream] Saving to: 'flink-1.7.1-bin-scala_2.11.tgz' 22%[===>] 55.95M 7.88MB/s eta 32s

Now, uncompress the tar file.

```
tar -xzf flink-1.7.1-bin-scala_2.11.tgz
```

```
ubuntu@ubuntu-VirtualBox:~$ tar -xzf flink-1.7.1-bin-scala_2.11.tgz
ubuntu@ubuntu-VirtualBox:~$ ls
Desktop examples.desktop Music Templates
Documents flink-1.7.1 Pictures Videos
Downloads flink-1.7.1-bin-scala_2.11.tgz Public
ubuntu@ubuntu-VirtualBox:~$
```

Go to Flink's home directory.

```
cd flink-1.7.1/
```



Start the Flink Cluster.

```
./bin/start-cluster.sh
```

```
wbuntu@ubuntu-VirtualBox: ~/flink-1.7.1
ubuntu@ubuntu-VirtualBox: ~/ flink-1.7.1/
ubuntu@ubuntu-VirtualBox: ~/flink-1.7.1$ ./bin/start-cluster.sh
Starting cluster.
Starting standalonesession daemon on host ubuntu-VirtualBox.
Starting taskexecutor daemon on host ubuntu-VirtualBox.
ubuntu@ubuntu-VirtualBox: ~/flink-1.7.1$
```

Open the Mozilla browser and go to the below URL, it will open the Flink Web Dashboard.

http://localhost:8081

This is how the User Interface of Apache Flink Dashboard looks like.

🚰 Apache Flink Web D × 💠													
(i localhost:8081/#/overview					C	Q Search		☆	Ê	+	^	9 🛡	≡
🗞 Apache Flink Dashboard	Overview	Version: 1.7.1	Commit: 89eaft	04									
an Overview				0		Total Jobs							
➡ Running Jobs				Task Managers		Running						0	
Completed Jobs				0		Finished						0	
击 Task Managers				Task Slots		Canceled						0	
琶 Job Manager				0		Failed						0	
± Submit new Job				Available Task Slots									
	Running Jo	obs											
	Start Time	End	I Time	Duration	Job I	Name	Job ID	Task	KS		Status		-
	Completed	I Jobs											
	Start Time	Enc	I Time	Duration	Job I	Name	Job ID	Task	(S		Status		

Now the Flink cluster is up and running.



Flink has a rich set of APIs using which developers can perform transformations on both batch and real-time data. A variety of transformations includes mapping, filtering, sorting, joining, grouping and aggregating. These transformations by Apache Flink are performed on distributed data. Let us discuss the different APIs Apache Flink offers.

Dataset API

Dataset API in Apache Flink is used to perform batch operations on the data over a period. This API can be used in Java, Scala and Python. It can apply different kinds of transformations on the datasets like filtering, mapping, aggregating, joining and grouping.

Datasets are created from sources like local files or by reading a file from a particular sourse and the result data can be written on different sinks like distributed files or command line terminal. This API is supported by both Java and Scala programming languages.

Here is a Wordcount program of Dataset API:

```
public class WordCountProg {
    public static void main(String[] args) throws Exception {
        final ExecutionEnvironment env =
ExecutionEnvironment.getExecutionEnvironment();
        DataSet<String> text = env.fromElements(
            "Hello",
            "My Dataset API Flink Program");
        DataSet<Tuple2<String, Integer>> wordCounts = text
            .flatMap(new LineSplitter())
            .groupBy(0)
            .sum(1);
        wordCounts.print();
    }
    public static class LineSplitter implements FlatMapFunction<String,
Tuple2<String, Integer>> {
        @Override
        public void flatMap(String line, Collector<Tuple2<String, Integer>>
out) {
```



```
for (String word : line.split(" ")) {
    out.collect(new Tuple2<String, Integer>(word, 1));
  }
}
```

DataStream API

}

This API is used for handling data in continuous stream. You can perform various operations like filtering, mapping, windowing, aggregating on the stream data. There are various sources on this data stream like message queues, files, socket streams and the result data can be written on different sinks like command line terminal. Both Java and Scala programming languages support this API.

Here is a streaming Wordcount program of DataStream API, where you have continuous stream of word counts and the data is grouped in the second window.

```
import org.apache.flink.api.common.functions.FlatMapFunction;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.windowing.time.Time;
import org.apache.flink.util.Collector;
public class WindowWordCountProg {
    public static void main(String[] args) throws Exception {
        StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
        DataStream<Tuple2<String, Integer>> dataStream = env
                .socketTextStream("localhost", 9999)
                .flatMap(new Splitter())
                .keyBy(0)
                .timeWindow(Time.seconds(5))
                .sum(1);
```

```
dataStream.print();
```



```
env.execute("Streaming WordCount Example");
}
public static class Splitter implements FlatMapFunction<String,
Tuple2<String, Integer>> {
    @Override
    public void flatMap(String sentence, Collector<Tuple2<String, Integer>>
out) throws Exception {
    for (String word: sentence.split(" ")) {
        out.collect(new Tuple2<String, Integer>(word, 1));
        }
    }
}
```



Table API is a relational API with SQL like expression language. This API can do both batch and stream processing. It can be embedded with Java and Scala Dataset and Datastream APIs. You can create tables from existing Datasets and Datastreams or from external data sources. Through this relational API, you can perform operations like join, aggregate, select and filter. Whether the input is batch or stream, the semantics of the query remains the same.

Here is a sample Table API program:

```
// for batch programs use ExecutionEnvironment instead of
StreamExecutionEnvironment
val env = StreamExecutionEnvironment.getExecutionEnvironment
// create a TableEnvironment
val tableEnv = TableEnvironment.getTableEnvironment(env)
// register a Table
tableEnv.registerTable("table1", ...)
                                       // or
tableEnv.registerTableSource("table2", ...)
                                              // or
tableEnv.registerExternalCatalog("extCat", ...)
// register an output Table
tableEnv.registerTableSink("outputTable", ...);
// create a Table from a Table API query
val tapiResult = tableEnv.scan("table1").select(...)
// Create a Table from a SQL query
val sqlResult = tableEnv.sqlQuery("SELECT ... FROM table2 ...")
// emit a Table API result Table to a TableSink, same for SQL result
tapiResult.insertInto("outputTable")
// execute
env.execute()
```



9. Apache Flink — Creating a Flink Application

In this chapter, we will learn how to create a Flink application.

Open Eclipse IDE, click on New Project and Select Java Project.

🛛 🖨 🖪 Resource - Eclipse	😵 🗉 New Project]
] 📬 ▾ 🔛 🕼 👜] 💁 ▾	Select a wizard Create a Java project	Resource
E	Wizards: type filter text	
	 # Java Project from Existing Ant Buildfile Plug-in Project General CVS Dava 	
ि Outline & An outline is not available.	Plug-in Development	
	Cancel Finish	ath Location

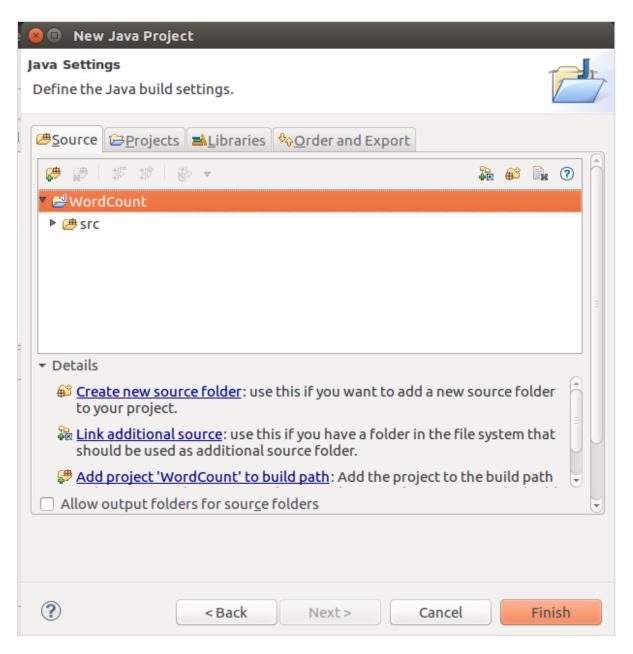


Give Project Name and click on Finish.

😣 🗉 New Java Project										
Create a Java Project										
Create a Java project in the workspace or in an e	xternal location.									
Project name: WordCount										
Use <u>d</u> efault location										
Location: /home/ubuntu/workspace/WordCo	unt B <u>r</u> owse									
JRE										
O Use an execution environment JRE:	OSGi/Minimum-1.2									
Use a project specific JRE:	java-8-oracle 🗘									
O Use default JRE (currently 'java-8-oracle')	Configure JREs									
Designt laugut										
Project layout										
 <u>U</u>se project folder as root for sources and 	class files									
<u>Create separate folders for sources and classified in the separate folders for sources and classified in the separate separat</u>	ass files Configure default									
· · · ·										
? < Back Next >	Cancel Finish									



Now, click on Finish as shown in the following screenshot.





😣 🖨 🗉 🛛 Java - Eclipse Platform					
] 📬 🔹 🖩 🖨 📄 🌞 🔹 🗿 💌	♀ ▼] ♥] ♥ ♂ ▼]	ت 😒 🖋 🕈	{} ▼ \$\$ \$ ▼ \$	∇	🗈 🖏 Java 🖒 F "
🗏 Package Explorer 🛛 🗧 🖥	New	•	Java Project		🗄 Outline 🛛 🦳 🗖 🗖
□ 🔄 ▽	Open in New Window		P <u>r</u> oject		An outline is not available.
▼ 😂 WordCount	Open Type Hierarchy	F4	Package		
🗁 src	Show In	Shift+Alt+W →	Class		
JRE System Library [java-8-or	Copy Copy Qualified Name Paste Delete	Ctrl+C Ctrl+V Delete	Interface Enum Annotation Source Folder		
	Build Path Source Refactor	→ Shift+Alt+S Shift+Alt+T	Java Working Set Folder File Untitled Text File		
	Import Export		JUnit Test Case <u>O</u> ther	Ctrl+N	
	Refresh Assign Working Sets	F5			

Now, right-click on **src** and go to New >> Class.



Give a class name and click on Finish.

😣 🗉 New Java Class											
Java Class			0								
A The use of the del	A The use of the default package is discouraged.										
Source folder:	WordCount/src		Browse								
Package:		(default)	Browse								
Enclosing type:			Browse								
Name:	WordCount										
Modifiers:	opublic	protected									
	□ abstract □ final □ static										
Superclass:	java.lang.Object		Browse								
Interfaces:			Add								
Which method stub	s would you like to create?										
	public static void main(String[] args)										
	Constructors from superclass										
	🧭 Inherited abstract methods										
?		Cancel	Finish								

Copy and paste the below code in the Editor.

<pre>import org.apache.flink.api.common.functions.FlatMapFunction;</pre>
<pre>import org.apache.flink.api.java.DataSet;</pre>
<pre>import org.apache.flink.api.java.ExecutionEnvironment;</pre>
<pre>import org.apache.flink.api.java.tuple.Tuple2;</pre>
<pre>import org.apache.flink.api.java.utils.ParameterTool;</pre>
<pre>import org.apache.flink.util.Collector;</pre>
<pre>public class WordCount {</pre>



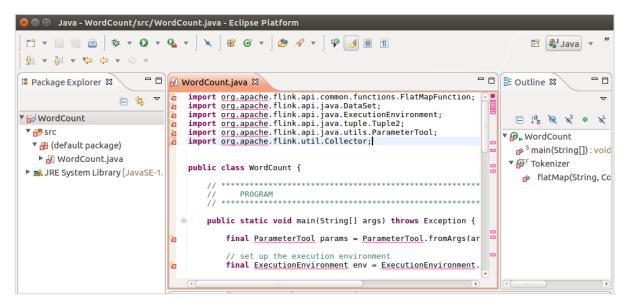
```
//
  *****
                11
           PROGRAM
     11
                         public static void main(String[] args) throws Exception {
          final ParameterTool params = ParameterTool.fromArgs(args);
          // set up the execution environment
          final ExecutionEnvironment env =
ExecutionEnvironment.getExecutionEnvironment();
          // make parameters available in the web interface
           env.getConfig().setGlobalJobParameters(params);
          // get input data
          DataSet<String> text = env.readTextFile(params.get("input"));
          DataSet<Tuple2<String, Integer>> counts =
                       // split up the lines in pairs (2-tuples) containing:
(word,1)
                       text.flatMap(new Tokenizer())
                       // group by the tuple field "0" and sum up tuple field
"1"
                       .groupBy(0)
                       .sum(1);
          // emit result
          if (params.has("output")) {
                 counts.writeAsCsv(params.get("output"), "\n", " ");
                 // execute program
                 env.execute("WordCount Example");
           } else {
                 System.out.println("Printing result to stdout. Use --output
to specify output path.");
                 counts.print();
                                                                         <del>19</del>
```

tutoria

```
}
    }
    //
                         *******
    //
          USER FUNCTIONS
    //
         public static final class Tokenizer implements FlatMapFunction<String,</pre>
Tuple2<String, Integer>> {
          public void flatMap(String value, Collector<Tuple2<String,</pre>
Integer>> out) {
               // normalize and split the line
               String[] tokens = value.toLowerCase().split("\\W+");
               // emit the pairs
               for (String token : tokens) {
                     if (token.length() > 0) {
                          out.collect(new Tuple2<>(token, 1));
                     }
               }
          }
    }
}
```



You will get many errors in the editor, because Flink libraries need to be added to this project.



Right-click on the project >> Build Path >> Configure Build Path.

😣 🗩 🗉 🛛 Java - WordCount/src/WordCount.java - Eclipse Platform									
] <mark>□ </mark>	New Go Into	Þ] 🥭 🛷 🔻] 🍄 🦽 🔳	Π					
Package	Open in New Window Open Type Hierarchy Show In	F4 Shift+Alt+W →	e.flink.api.common.funct e.flink.api.java.DataSet	ions.FlatMapFunction;					
▼ 🥵 WordC ▼ 🥭 src ▼ 🖶 (def ▶ 💽 W	Copy Copy Qualified Name Paste Delete	Ctrl+C Ctrl+V Delete	e.flink.api.java.Executi e.flink.api.java.aggrega e.flink.api.java.tuple.T e.flink.api.java.utils.P e.flink.util.Collector;	onEnvironment; tion.Aggregations; uple2;					
▶ 🛋 JRE S	Build Path Source Refactor	Shift+Alt+S → Shift+Alt+T →	Link <u>S</u> ource <u>N</u> ew Source Folder	throws Exception { ExecutionEnvironment.					
	Import Export	ShirtrAttri	<u>U</u> se as Source Folder Add External Archi <u>v</u> es Add <u>L</u> ibraries	<pre>inclus(io! Who's there?"); vordCounts = text</pre>					
	Refresh Close Project Assign Working Sets	F5	<u>C</u> onfigure Build Path I(1) ;						



😣 🗉 Properties for WordCount Java Build Path × type filter text () v () v v Resource Bource
 Gerojects
 ALibraries
 ふOrder and Export Builders JARs and class folders on the build path: Java Build Path JRE System Library [java-8-oracle] Java Code Style Add JARs... Java Compiler Add External JARs... Java Editor Javadoc Location Add Variable... Project References Add Library... Run/Debug Settings Add Class Folder... Add External Class Folder...

Select the Libraries tab and click on Add External JARs.

Go to Flink's lib directory, select all the 4 libraries and click on OK.

😣 🗈 JAR Selection				
	flink-1.7.1 lib			
Places	Name	۵	Size	Modified
🔍 Search	🌁 flink-dist_2.11-1.7.1.jar		93.2 MB	Saturday 15 December 2018
Recently Used	🔤 flink-python_2.11-1.7.1.jar		141.9 kB	Saturday 15 December 2018
📠 ubuntu	🌁 log4j-1.2.17.jar			Thursday 01 March 2018
🛅 Desktop	🔤 slf4j-log4j12-1.7.15.jar		9.9 kB	Thursday 13 December 2018
File System				
OVBox_GAs_6.0.2				
Documents				
🚺 Music				
间 Pictures				
Videos				
Downloads				
+ -				*.jar;*.zip ‡
				Cancel OK



Go to the Order and Export tab, select all the libraries and click on OK.

😣 🗉 Properties for Wo	rdCount	
type filter text	Java Build Path	▼ ▼
 Resource Builders Java Build Path Java Code Style 	 ➢ Source ➢ Projects ➢ Libraries ➢ Order and Export Build class path order and exported entries: (Exported entries are contributed to dependent projects) 	
Java Compiler	Count/src	Up
 Java Editor Javadoc Location 	 ✓ ➡ JRE System Library [java-8-oracle] ✓ ➡ flink-dist_2.11-1.7.1.jar - /home/ubuntu/flink-1.7.1/lib 	Down
Project References	 Flink-python_2.11-1.7.1.jar - /home/ubuntu/flink-1.7.1/l Gog4j-1.2.17.jar - /home/ubuntu/flink-1.7.1/lib 	Тор
Run/Debug Settings	Slf4j-log4j12-1.7.15.jar - /home/ubuntu/flink-1.7.1/lib	Bottom
		Select All Deselect All
?	Can	сel ОК

You will see that the errors are no more there.



😣 🖨 🗊 🛛 Java - Word	Count/src/WordCount.jav	a - Eclipse Platform	1
] 🗗 🔻 🔛 🕼 🚔 🛛] 설 👻 🏹 👻 🍫 🤇	Ne <u>w</u> Go <u>I</u> nto	Þ	? ▼] ♀ ⊿ ■ 1
Package Explorer	Open in <u>N</u> ew Window Ope <u>n</u> Type Hierarchy Sho <u>w</u> In	F4 Shift+Alt+W →	.api.common.functic .api.java.DataSet;
▼ 😅 WordCount ▼ 进 src ▼ 进 (default pack ▶ 🔊 WordCount	<u>C</u> opy Copy Qualified Name <u>P</u> aste <u>D</u> elete	Ctrl+C Ctrl+V Delete	
 ▶ ➡ JRE System Lib ▶ ➡ Referenced Lib 	<u>B</u> uild Path <u>S</u> ource Refac <u>t</u> or	Shift+Alt+S > Shift+Alt+T >	************************************
	<u>I</u> mport Exp <u>o</u> rt		nain(String[] args) Fool params = Paran
	Re <u>f</u> resh Clo <u>s</u> e Project <u>A</u> ssign Working Sets		<pre>kecution environmer Environment env = E</pre>

Now, let us export this application. Right-click on the project and click on Export.



Select JAR file and click Next>

😣 🗉 Export						
Select				X		
Export resources into a JAR file on the local file system.						
Select an export dest	ination:					
type filter text				×		
🕨 🗁 General						
🕨 🗁 Install						
🔻 🗁 Java						
📮 JAR file						
🔍 Javadoc						
🖏 Runnable JAR						
🕨 🗁 Plug-in Develop	ment					
🕨 🗁 Run/Debug						
🕨 🗁 Team						
?	< Back	Next >	Cancel	Finish		



Give a destination path and click on Next>

😣 🗉 JAR Export					
JAR File Specification	—				
Define which resources should be exported	into the JAR.				
Select the resources to <u>e</u> xport:					
🕨 🛑 🔛 WordCount	🕑 🗎 .classpath				
	🗹 📄 .project				
Export generated <u>c</u> lass files and resour	ces				
Export all output folders for checked pr	ojects				
Export Java <u>s</u> ource files and resources					
Export refactorings for checked project	s. Select refactorings				
Select the export destination:					
JAR file: /home/ubuntu/wordcount.jar	▼ B <u>r</u> owse				
Options:					
Compress the contents of the JAR file					
Add directory entries					
Overwrite existing files without warning					
? < Back N	ext > Cancel Finish				



Click on Next>

😣 🗊 JAR Export	
JAR Packaging Options	7
Define the options for the JAR export.	
Select options for handling problems:	
Export class files with compile errors	
Export class files with compile warnings	
Create source folder structure	
Build projects if not built automatically	
Save the description of this JAR in the workspace	
Description file:	B <u>r</u> owse
? < Back Next > Cancel	Finish



Click on Browse, select the main class (WordCount) and click Finish.

😣 💿 JAR Export				
JAR Manifest Specifica Customize the manifes		R file.		
Specify the manifest:				
🖲 Generate the mani	fest file			
Save the manife	st in the works	pace		
Use the saved m	anifest in the g	enerated JAR de	escription file	
Manifest file:				Browse
○ Use existing manif	est from works	pace		
Manifest file:				Browse
Seal contents:				
Seal the JAR				Details
Seal some package	S		Nothing sealed	Details
Select the class of the a	pplication entr	y point:		
Main class: WordCou	nt			Browse
?	< Back	Next >	Cancel	Finish

Note: Click OK, in case you get any warning.



Run the below command. It will further run the Flink application you just created.

./bin/flink run /home/ubuntu/wordcount.jar --input README.txt --output
/home/ubuntu/output

ubuntu@ubuntu-VirtualBox:~/flink-1.7.1\$./bin/flink run /home/ubuntu/wordcount.jar --input README.txt --output /home/ ubuntu/output Starting execution of program Program execution finished Job with JobID 1abb8d30a2c898fa50e7ee997a73e431 has finished. Job Runtime: 805 ms ubuntu@ubuntu-VirtualBox:~/flink-1.7.1\$



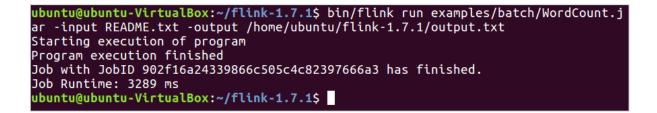
10. Apache Flink — Running a Flink Program

In this chapter, we will learn how to run a Flink program.

Let us run the Flink wordcount example on a Flink cluster.

Go to Flink's home directory and run the below command in the terminal.

bin/flink run examples/batch/WordCount.jar -input README.txt -output /home/ubuntu/flink-1.7.1/output.txt



Go to Flink dashboard, you will be able to see a completed job with its details.

🖀 Apache Flink Web Dashl 🗙 🕂								
$\overleftarrow{\leftarrow}$ \rightarrow \overleftarrow{C}	(i) localhos	t:8081/#/completed-	jobs		🚥 🔽 🗘 Search			
è	Completed Job	DS						
Overview	Start Time	End Time	Duration	Job Name	Job ID	Tasks	Status	
■ Running Jobs	2019-01-25, 16:29:40	2019-01-25, 16:29:44	3s	WordCount Example	902f16a24339866c505c4c82397666a3	3 0 0 0 0 3 0 0 0 0	FINISHED	
Completed Jobs								

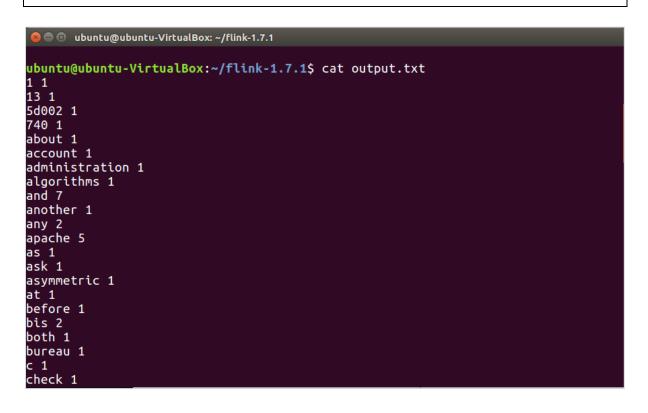


WordCount Ex 902f16a24339866	xample c505c4c82397666a3	2	019-01-25, 16:29	:40 - 2019-01-25, 16	:29:44	3s	0 3 0	0 0 0 0	0			
Overview	Timeline Ex	ceptions	Configuratio	on								
	DataSour java:66 java.i > -> Fit in(W gt; Comb	ce (at main) (org.apacl io.TextInput atMap (Flatl ordCount.ja -&	(Format)) Map at ma ava:76))), at main(W a:79) 1	Hash Paritiion on [0], Sort (combining) on [0:A5	Reduce (S	Group Red SUM(1), at r unt.java.7 Parallelism: ion: Sorted Gro	main(Word '9)	ICo Forwa	— h:.	Data Si aSink (CsvOutp /home/ubuntu// tput.txt, dellin Parallelien Operation: (utFormat (pat link-1.7.1/ou niter:)) n: 1	-
Subtasks	Task Metrics	Water	marks Accun	nulators Checkp	oints							
Aggregate task statistics by TaskManager												
Start Time	End Time	Duration	Name			Byte		Records received	Bytes sent	Records sent	Parallelism	Т
2019-01-25,	2019-01-25,	1s	CHAIN DataSo	,		0 B		0	0 B	111	1	0

If you click on Completed Jobs, you will get detailed overview of the jobs.

To check the output of wordcount program, run the below command in the terminal.

cat output.txt





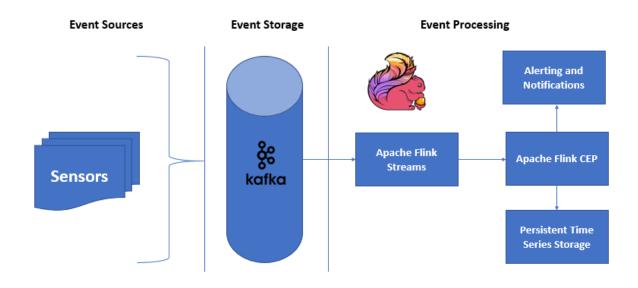
11. Apache Flink — Libraries

In this chapter, we will learn about the different libraries of Apache Flink.

Complex Event Processing (CEP)

FlinkCEP is an API in Apache Flink, which analyses event patterns on continuous streaming data. These events are near real time, which have high throughput and low latency. This API is used mostly on Sensor data, which come in real-time and are very complex to process.

CEP analyses the pattern of the input stream and gives the result very soon. It has the ability to provide real-time notifications and alerts in case the event pattern is complex. FlinkCEP can connect to different kind of input sources and analyse patterns in them.



This how a sample architecture with CEP looks like:

Sensor data will be coming in from different sources, Kafka will act as a distributed messaging framework, which will distribute the streams to Apache Flink, and FlinkCEP will analyse the complex event patterns.

You can write programs in Apache Flink for complex event processing using Pattern API. It allows you to decide the event patterns to detect from the continuous stream data. Below are some of the most commonly used CEP patterns:

Begin

It is used to define the starting state. The following program shows how it is defined in a Flink program:

```
Pattern<Event, ?> next = start.next("next");
```



Where

It is used to define a filter condition in the current state.

```
patternState.where(new FilterFunction <Event>() {
@Override
public boolean filter(Event value) throws Exception {
return ... // some condition
}
});
```

Next

It is used to append a new pattern state and the matching event needed to pass the previous pattern.

Pattern<Event, ?> next = start.next("next");

FollowedBy

It is used to append a new pattern state but here other events can occur b/w two matching events.

Pattern<Event, ?> followedBy = start.followedBy("next");

Gelly

Apache Flink's Graph API is Gelly. Gelly is used to perform graph analysis on Flink applications using a set of methods and utilities. You can analyse huge graphs using Apache Flink API in a distributed fashion with Gelly. There are other graph libraries also like Apache Giraph for the same purpose, but since Gelly is used on top of Apache Flink, it uses single API. This is very helpful from development and operation point of view.

Let us run an example using Apache Flink API – Gelly.

Firstly, you need to copy 2 Gelly jar files from opt directory of Apache Flink to its lib directory. Then run flink-gelly-examples jar.

```
cp opt/flink-gelly* lib/
```

./bin/flink run examples/gelly/flink-gelly-examples_*.jar





Let us now run the PageRank example.

PageRank computes a per-vertex score, which is the sum of PageRank scores transmitted over in-edges. Each vertex's score is divided evenly among out-edges. High-scoring vertices are linked to by other high-scoring vertices.

The result contains the vertex ID and the PageRank score.

usage: flink run examples/flink-gelly-examples_<version>.jar --algorithm
PageRank [algorithm options] --input <input> [input options] --output <output>
[output options]
./bin/flink run examples/gelly/flink-gelly-examples_*.jar --algorithm PageRank
--input CycleGraph --vertex_count 2 --output Print





Apache Flink's Machine Learning library is called FlinkML. Since usage of machine learning has been increasing exponentially over the last 5 years, Flink community decided to add this machine learning APO also in its ecosystem. The list of contributors and algorithms are increasing in FlinkML. This API is not a part of binary distribution yet.

Here is an example of linear regression using FlinkML:

```
// LabeledVector is a feature vector with a label (class or real value)
val trainingData: DataSet[LabeledVector] = ...
val testingData: DataSet[Vector] = ...
// Alternatively, a Splitter is used to break up a DataSet into training and
testing data.
val dataSet: DataSet[LabeledVector] = ...
val trainTestData: DataSet[TrainTestDataSet] = Splitter.trainTestSplit(dataSet)
val trainingData: DataSet[LabeledVector] = trainTestData.training
val testingData: DataSet[Vector] = trainTestData.testing.map(lv => lv.vector)
val mlr = MultipleLinearRegression()
  .setStepsize(1.0)
  .setIterations(100)
  .setConvergenceThreshold(0.001)
mlr.fit(trainingData)
// The fitted model can now be used to make predictions
val predictions: DataSet[LabeledVector] = mlr.predict(testingData)
```

Inside **flink-1.7.1/examples/batch/** path, you will find KMeans.jar file. Let us run this sample FlinkML example.

This example program is run using the default point and the centroid data set.

```
./bin/flink run examples/batch/KMeans.jar --output Print
```



ubuntu@ubuntu-VirtualBox:~/flink-1.7.1\$./bin/flink run examples/batch/KMeans.jar --output Print
Starting execution of program
Executing K-Means example with default point data set.
Use --points to specify file input.
Executing K-Means example with default centroid data set.
Use --centroids to specify file input.
Program execution finished
Job with JobID 6619346b85610635c21e96b16d4f7dfc has finished.
Job Runtime: 2373 ms



In this chapter, we will understand a few test cases in Apache Flink.

Apache Flink – Bouygues Telecom

Bouygues Telecom is one of the largest telecom organization in France. It has 11+ million mobile subscribers and 2.5+ million fixed customers. Bouygues heard about Apache Flink for the first time in a Hadoop Group Meeting held at Paris. Since then they have been using Flink for multiple use-cases. They have been processing billions of messages in a day in real-time through Apache Flink.

This is what Bouygues has to say about Apache Flink: "We ended up with Flink because the system supports true streaming - both at the API and at the runtime level, giving us the programmability and low latency that we were looking for. In addition, we were able to get our system up and running with Flink in a fraction of the time compared to other solutions, which resulted in more available developer resources for expanding the business logic in the system."

At Bouygues, customer experience is the highest priority. They analyse data in real-time so that they can give below insights to their engineers:

- Real-Time Customer Experience over their network
- What is happening globally on the network
- Network evaluations and operations

They created a system called LUX (Logged User Experience) which processed massive log data from network equipment with internal data reference to give quality of experience indicators which will log their customer experience and build an alarming functionality to detect any failure in consumption of data within 60 seconds.

To achieve this, they needed a framework which can take massive data in real-time, is easy to set up and provides rich set of APIs for processing the streamed data. Apache Flink was a perfect fit for Bouygues Telecom.

Apache Flink – Alibaba

Alibaba is the largest ecommerce retail company in the world with 394 billion \$ revenue in 2015. Alibaba search is the entry point to all the customers, which shows all the search and recommends accordingly.

Alibaba uses Apache Flink in its search engine to show results in real-time with highest accuracy and relevancy for each user.

Alibaba was looking for a framework, which was:

- Very Agile in maintaining one codebase for their entire search infrastructure process.
- Provides low latency for the availability changes in the products on the website.



• Consistent and cost effective.

Apache Flink qualified for all the above requirements. They need a framework, which has a single processing engine and can process both batch and stream data with same engine and that is what Apache Flink does.

They also use Blink, a forked version for Flink to meet some unique requirements for their search. They are also using Apache Flink's Table API with few improvements for their search.

This is what Alibaba had to say about apache Flink: "Looking back, it was no doubt a huge year for Blink and Flink at Alibaba. No one thought that we would make this much progress in a year, and we are very grateful to all the people who helped us in the community. Flink is proven to work at the very large scale. We are more committed than ever to continue our work with the community to move Flink forward!"



14. Apache Flink — Flink vs Spark vs Hadoop

Here is a comprehensive table, which shows the comparison between three most popular big data frameworks: Apache Flink, Apache Spark and Apache Hadoop.

	Apache Hadoop	Apache Spark	Apache Flink		
Year of Origin	2005	2009	2009		
Place of Origin	MapReduce (Google) Hadoop (Yahoo)	University of California, Berkeley	Technical University of Berlin		
Data Processing Engine	Batch	Batch	Stream		
Processing Speed	Slower than Spark and Flink	100x Faster than Hadoop	Faster than spark		
Programming Languages	Java, C, C++, Ruby, Groovy, Perl, Python	Java, Scala, python and R	Java and Scala		
Programming Model	MapReduce	Resilient distributed Datasets (RDD)	Cyclic dataflows		
Data Transfer	Batch	Batch	Pipelined and Batch		
Memory Management	Disk Based	JVM Managed	Active Managed		
Latency	Low	Medium	Low		
Throughput	Medium	High	High		
Optimization	Manual	Manual	Automatic		
API	Low-level	High-level	High-level		
Streaming Support	NA	Spark Streaming	Flink Streaming		
SQL Support	Hive, Impala	SparkSQL	Table API and SQL		
Graph Support	NA	GraphX	Gelly		
Machine Learning Support	NA	SparkML	FlinkML		



The comparison table that we saw in the previous chapter concludes the pointers pretty much. Apache Flink is the most suited framework for real-time processing and use cases. Its single engine system is unique which can process both batch and streaming data with different APIs like Dataset and DataStream.

It does not mean Hadoop and Spark are out of the game, the selection of the most suited big data framework always depends and vary from use case to use case. There can be several use cases where a combination of Hadoop and Flink or Spark and Flink might be suited.

Nevertheless, Flink is the best framework for real time processing currently. The growth of Apache Flink has been amazing and the number of contributors to its community is growing day by day.

Happy Flinking!

