Introduction to Apache Flink™

Maximilian Michels
mxm@apache.org
@stadtlegende

Ufuk Celebi
uce@apache.org
@iamucae
About Us

- Ufuk Celebi <uce@apache.org>
- Maximilian Michels <mxm@apache.org>

- Apache Flink Committers

- Working full-time on Apache Flink at dataArtisans
Goal: Get you started with Apache Flink

- Overview
- Internals
- APIs
- Exercise
- Demo
- Advanced features
Exercises

- We will ask you to do an exercise using Flink

- Please check out the website for this session: http://dataArtisans.github.io/eit-summerschool-15

- The website also contains a solution which we will present later on, but, first, try to solve the exercise.
1 year of Flink - code

April 2014

Stratosphere accepted as Apache Incubator Project
16 Apr 2014
We are happy to announce that Stratosphere has been accepted as a project for the Apache incubator. The proposal has been accepted by the Incubator PMC members earlier this week. The Apache Incubator is the first step in the process of giving a project to the Apache Software Foundation. While under incubation, the project will move to the Apache infrastructure and adopt the community-driven development principles of the Apache Foundation. Projects can graduate from incubation to become top-level projects if they show activity, a healthy community dynamic, and releases.

We are glad to have Alan Gates as champion on board as well as a set of great mentors, including Sean Owen, Ted Dunning, Owen O’Riley, Henry Sapienza, and Ashutosh Chauhan. We are confident that we will make this a great open source effort.

April 2015

Hadoop M/R
Flink core
Local
Remote
Yarn

DataSet API (Java/Scala)
Gelly
Table
ML
DataStream (Java/Scala)
Dataflow
MRQL
Table
SAMOA
Dataflow
In top 5 of Apache's big data projects after one year in the Apache Software Foundation.
The Apache Way

- Independent, non-profit organization
- Community-driven open source software development approach
- Consensus-based decision making
- Public communication and open to new contributors

The Apache Software Foundation
Community-led development since 1999.
Integration into the Big Data Stack

Programming APIs
- Java
- Scala
- Hadoop MapReduce

Data Access
- HDFS
- Azure Storage
- TACHYON
- Amazon S3
- NFS
- Apache HBase
- Google Cloud Storage
- Kafka

Deployment
- Direct
- Hadoop YARN
- Apache Tez
- Cloud
A stream processor with many applications

Streaming dataflow runtime
What can I do with Flink?
Basic API Concept

How do I write a Flink program?
1. Bootstrap sources
2. Apply operations
3. Output to source
Stream & Batch Processing

- **DataStream API**

<table>
<thead>
<tr>
<th>Stock Feed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>Microsoft</td>
</tr>
<tr>
<td>Google</td>
</tr>
<tr>
<td>Apple</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

- Alert if Microsoft > 120
- Sum every 10 seconds

- Write event to database
- Alert if sum > 10000

- **DataSet API**

<table>
<thead>
<tr>
<th>b</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Map
- Reduce

- a
- 12
- ...
## Streaming & Batch

<table>
<thead>
<tr>
<th></th>
<th>Streaming</th>
<th>Batch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>infinite</td>
<td>finite</td>
</tr>
<tr>
<td><strong>Data transfer</strong></td>
<td>pipelined</td>
<td>blocking or pipelined</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>low</td>
<td>high</td>
</tr>
</tbody>
</table>
Scaling out
Scaling up
Flink’s Architecture
Architecture Overview

- Client
- Master (Job Manager)
- Worker (Task Manager)
Client

- Optimize
- Construct job graph
- Pass job graph to job manager
- Retrieve job results

```scala
case class Path(from: Long, to: Long)
val tc = edges.iterate(10) {
  paths: DataSet[Path] =>
  val next = paths.
  .join(edges).
  .where("to")
  .equalTo("from")
  .map((path, edge) =>
    Path(path.from, edge.to)
  )
  .union(paths)
  .distinct()
  .next
}
```

Job Manager
Job Manager

- **Parallelization**: Create Execution Graph
- **Scheduling**: Assign tasks to task managers
- **State tracking**: Supervise the execution
Task Manager

- Operations are split up into tasks depending on the specified parallelism.
- Each parallel instance of an operation runs in a separate task slot.
- The scheduler may run several tasks from different operators in one task slot.
case class Path (from: Long, to: Long)
val tc = edges.iterate(10)
  paths: DataSet[Path] =>
  val next = paths
    .join(edges)
    .where("to")
    .equalTo("from")
    .map { (path, edge) =>
      Path(path.from, edge.to)
    }
    .distinct()
  next
}

From Program to Execution

Dataflow Graph

Optimizer

Type extraction stack

Pre-flight (Client)

Program

Dataflow metadata

Task scheduling

Job Manager

Task Managers

deploy operators

track intermediate results

Deploy (Job Manager)

Task Managers
Flink's execution model
Flink execution model

- A program is a graph (DAG) of operators
- Operators = computation + state
- Operators produce intermediate results = logical streams of records
- Other operators can consume those
A map-reduce job with Flink

"Blocked" result partition

TaskManager 1

TaskManager 2

M1

M2

RP1

R1

RP2

R2

JobManager

ExecutionGraph
Streaming

"Pipelined" result partition

Task Manager 1

Task Manager 2
Non-native streaming

while (true) {
  // get next few records
  // issue batch job
}
Batch on Streaming

- Batch programs are a special kind of streaming program

**Infinite Streams** ↔ **Finite Streams**

**Stream Windows** ↔ **Global View**

**Pipelined Data Exchange** ↔ **Pipelined or Blocking Exchange**

**Streaming Programs** ↔ **Batch Programs**
Stream processor applications

Stream processing

Batch processing

Machine Learning at scale

Graph Analysis
Demo
Introduction to the DataSet API
Flink’s APIs

DataStream (Java/Scala)

DataSet (Java/Scala)

Streaming dataflow runtime
case class Word (word: String, frequency: Int)

DataSet API (batch):

val lines: DataSet[String] = env.readTextFile(...)  
lines.flatMap {line => line.split(" ")  
  .map(word => Word(word,1))}  
  .groupBy("word").sum("frequency")  
  .print()

DataStream API (streaming):

val lines: DataStream[String] = env.fromSocketStream(...)  
lines.flatMap {line => line.split(" ")  
  .map(word => Word(word,1))}  
  .window(Time.of(5,SECONDS)).every(Time.of(1,SECONDS))  
  .groupBy("word").sum("frequency")  
  .print()
public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env =
            ExecutionEnvironment.getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts =
            // split up the lines in tuples containing: (word,1)
            inputText.flatMap(new Tokenizer())
            // group by the tuple field "0"
            .groupBy(0)
            // sum up tuple field "1"
            .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", " ");

    // execute program
    env.execute("WordCount Example");
}
public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env = 
        ExecutionEnvironment.getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts = 
        // split up the lines in tuples containing: (word,1)
        inputText.flatMap(new Tokenizer())
        // group by the tuple field "0"
            .groupBy(0)
        // sum up tuple field "1"
            .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", " ");
    // execute program
    env.execute("WordCount Example");
}
public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env =
        ExecutionEnvironment.getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts =
        inputText.flatMap(new Tokenizer())
        .groupBy(0)
        .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", " ");
    // execute program
    env.execute("WordCount Example");
}
public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env =
    ExecutionEnvironment.getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts =
    // split up the lines in tuples containing: (word,1)
    inputText.flatMap(new Tokenizer())
    // group by the tuple field "0"
    .groupBy(0)
    // sum up tuple field "1"
    .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", " ");
    // execute program
    env.execute("WordCount Example");
}
public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env =
            ExecutionEnvironment.getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts =
            // split up the lines in tuples containing: (word,1)
            inputTextflatMap(new Tokenizer())
            // group by the tuple field "0"
            .groupBy(0)
            // sum up tuple field "1"
            .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", " ");
    // execute program
    env.execute("WordCount Example");
}
User functions

public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env =
        ExecutionEnvironment.getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts =
        // split up the lines in tuples containing: (word,1)
        inputText.flatMap(new Tokenizer())
        // group by the tuple field "0"
        .groupBy(0)
        //sum up tuple field "1"
        .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", " ");
    // execute program
    env.execute("WordCount Example");
}
public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env = ExecutionEnvironment.getRuntimeInfo().getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts =
        // split up the lines in tuples containing: (word,1)
        inputText.flatMap(new Tokenizer())
        // group by the tuple field "0"
        .groupBy(0)
        // sum up tuple field "1"
        .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", "");
    // execute program
    env.execute("WordCount Example");
}
public static void main(String[] args) throws Exception {
    // set up the execution environment
    final ExecutionEnvironment env =
        ExecutionEnvironment.getExecutionEnvironment();

    // get input data either from file or use example data
    DataSet<String> inputText = env.readTextFile(args[0]);

    DataSet<Tuple2<String, Integer>> counts =
        inputText.flatMap(new Tokenizer())
            .groupBy(0)
            .reduceGroup(new SumWords());

    // emit result
    counts.writeAsCsv(args[1], "\n", " ");
    // execute program
    env.execute("WordCount Example");
}
public static class Tokenizer implements FlatMapFunction<String, Tuple2<String, Integer>> {

    @Override
    public void flatMap(String value, Collector<Tuple2<String, 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<String, Integer>(token, 1));
            }
        }
    }
}
public static class Tokenizer implements FlatMapFunction<String, Tuple2<String, Integer>> {

    @Override
    public void flatMap(String value,
                         Collector<Tuple2<String, 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<String, Integer>(token, 1));
            }
        }
    }
}
public static class Tokenizer
  implements FlatMapFunction<String, Tuple2<String, Integer>> {

    @Override
    public void flatMap(String value, Collector<Tuple2<String, 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<String, Integer>(token, 1));
        }
      }
    }
  }

public static class Tokenizer
    implements FlatMapFunction<String, Tuple2<String, Integer>> {

    @Override
    public void flatMap(String value, Collector<Tuple2<String, 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<String, Integer>(token, 1));
            }
        }
    }
}
public static class SumWords implements GroupReduceFunction<Tuple2<String, Integer>,
        Tuple2<String, Integer>> {

    @Override
    public void reduce(Iterable<Tuple2<String, Integer>> values,
            Collector<Tuple2<String, Integer>> out) {
        int count = 0;
        String word = null;
        for (Tuple2<String, Integer> tuple : values) {
            word = tuple.f0;
            count++;
        }
        out.collect(new Tuple2<String, Integer>(word, count));
    }
}
public static class SumWords implements GroupReduceFunction<Tuple2<String, Integer>,
   Tuple2<String, Integer>> {

   @Override
   public void reduce(Iterable<Tuple2<String, Integer>> values,
       Collector<Tuple2<String, Integer>> out) {

       int count = 0;
       String word = null;
       for (Tuple2<String, Integer> tuple : values) {
           word = tuple.f0;
           count++;
       }
       out.collect(new Tuple2<String, Integer>(word, count));
   }
}
public static class SumWords implements GroupReduceFunction<Tuple2<String, Integer>,
    Tuple2<String, Integer>> {

    @Override
    public void reduce(Iterable<Tuple2<String, Integer>> values,
                        Collector<Tuple2<String, Integer>> out) {

        int count = 0;
        String word = null;
        for (Tuple2<String, Integer> tuple : values) {
            word = tuple.f0;
            count++;
        }
        out.collect(new Tuple2<String, Integer>(word, count));
    }
}
public static class SumWords implements GroupReduceFunction<Tuple2<String, Integer>,
    Tuple2<String, Integer>> {

    @Override
    public void reduce(Iterable<Tuple2<String, Integer>> values,
        Collector<Tuple2<String, Integer>> out) {
        int count = 0;
        String word = null;
        for (Tuple2<String, Integer> tuple : values) {
            word = tuple.f0;
            count++;
        }
        out.collect(new Tuple2<String, Integer>(word, count));
    }
}
DataSet API Concepts
Data Types

- Basic Java Types
  - String, Long, Integer, Boolean,…
  - Arrays

- Composite Types
  - Tuple
  - PoJo (Java objects)
  - Custom type
Tuples

- The easiest, lightweight, and generic way of encapsulating data in Flink
- Tuple1 up to Tuple25

Tuple3<String, String, Integer> person =
    new Tuple3<>("Max", "Mustermann", 42);

// zero based index!
String firstName = person.f0;
String secondName = person.f1;
Integer age = person.f2;
Transformations: Map

```java
DataSet<Integer> integers = env.fromElements(1, 2, 3, 4);

// Regular Map - Takes one element and produces one element
DataSet<Integer> doubleIntegers =
    integers.map(new MapFunction<Integer, Integer>() {
        @Override
        public Integer map(Integer value) {
            return value * 2;
        }
    });

doubleIntegers.print();
> 2, 4, 6, 8

// Flat Map - Takes one element and produces zero, one, or more elements.
DataSet<Integer> doubleIntegers2 =
    integers.flatMap(new FlatMapFunction<Integer, Integer>() {
        @Override
        public void flatMap(Integer value, Collector<Integer> out) {
            out.collect(value * 2);
        }
    });

doubleIntegers2.print();
> 2, 4, 6, 8
```
Transformations: Filter

// The DataSet
DataSet<Integer> integers = env.fromElements(1, 2, 3, 4);

DataSet<Integer> filtered =

    integers.filter(new FilterFunction<Integer>() {
        @Override
        public boolean filter(Integer value) {
            return value != 3;
        }
    });

integers.print();
> 1, 2, 4
Groupings and Reduce

- DataSets can be split into groups
- Groups are defined using a common key

// (name, age) of employees
DataSet<Tuple2<String, Integer>> employees = …

// group by second field (age)
DataSet<Integer, Integer> grouped = employees.groupBy(1)
    // return a list of age groups with its counts
    .reduceGroup(new CountSameAge());
public static class CountSameAge implements GroupReduceFunction <Tuple2<String, Integer>, Tuple2<Integer, Integer>> {

    @Override
    public void reduce(Iterable<Tuple2<String, Integer>> values,
                        Collector<Tuple2<Integer, Integer>> out) {

        Integer ageGroup = 0;
        Integer countsInGroup = 0;

        for (Tuple2<String, Integer> person : values) {
            ageGroup = person.f1;
            countsInGroup++;
        }

        out.collect(new Tuple2<Integer, Integer>(ageGroup, countsInGroup));
    }
}
Joining two DataSets

<table>
<thead>
<tr>
<th>Authors</th>
<th>Posts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Name</td>
</tr>
<tr>
<td>1</td>
<td>Fabian</td>
</tr>
<tr>
<td>2</td>
<td>Julia</td>
</tr>
<tr>
<td>3</td>
<td>Max</td>
</tr>
<tr>
<td>4</td>
<td>Romeo</td>
</tr>
</tbody>
</table>

// authors (id, name, email)
DataSet<Tuple3<Integer, String, String>> authors = ..;
// posts (title, content, author_id)
DataSet<Tuple3<String, String, Integer>> posts = ..;

DataSet<Tuple2<
    Tuple3<Integer, String, String>,
    Tuple3<String, String, Integer>>
>> archive = authors.join(posts).where(0).equalTo(2);
Joining two DataSets

// authors (id, name, email)
DataSet<Tuple3<Integer, String, String>> authors = ..;

// posts (title, content, author_id)
DataSet<Tuple3<String, String, Integer>> posts = ..;

DataSet<Tuple2<Tuple3<Integer, String, String>, Tuple3<String, String, Integer>>> archive = authors.join(posts).where(0).equalTo(2);

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>email</th>
<th>Title</th>
<th>Content</th>
<th>Author id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fabian</td>
<td>fabian@..</td>
<td>..</td>
<td>..</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Julia</td>
<td>julia@...</td>
<td>..</td>
<td>..</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Julia</td>
<td>julia@...</td>
<td>..</td>
<td>..</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Romeo</td>
<td>romeo@...</td>
<td>..</td>
<td>..</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>Romeo</td>
<td>romeo@.</td>
<td>..</td>
<td>..</td>
<td>4</td>
</tr>
</tbody>
</table>
Join with join function

```java
// authors (id, name, email)
DataSet<Tuple3<Integer, String, String>> authors = ..;

// posts (title, content, author_id)
DataSet<Tuple3<String, String, Integer>> posts = ..;

// (title, author name)
DataSet<Tuple2<String, String>> archive =
    authors.join(posts).where(0).equalTo(2)
    .with(new PostsByUser());

public static class PostsByUser implements JoinFunction<Tuple3<Integer, String, String>,
    Tuple3<String, String, Integer>,
    Tuple2<String, String>> {
    @Override
    public Tuple2<String, String> join(
        Tuple3<Integer, String, String> left,
        Tuple3<String, String, Integer> right) {
        return new Tuple2<String, String>(left.f1, right.f0);
    }
}
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Title</strong></td>
</tr>
<tr>
<td>Fabian</td>
<td>..</td>
</tr>
<tr>
<td>Julia</td>
<td>..</td>
</tr>
<tr>
<td>Romeo</td>
<td>..</td>
</tr>
</tbody>
</table>

Archive

<table>
<thead>
<tr>
<th>Name</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fabian</td>
<td>..</td>
</tr>
<tr>
<td>Julia</td>
<td>..</td>
</tr>
<tr>
<td>Romeo</td>
<td>..</td>
</tr>
</tbody>
</table>
Data Sources / Data Sinks
Data Sources

Text
- `readTextFile("/path/to/file")`

CSV
- `readCsvFile("/path/to/file")`

Collection
- `fromCollection(collection)`
- `fromElements(1,2,3,4,5)`
Data Sources: Collections

ExecutionEnvironment env =
    ExecutionEnvironment.getExecutionEnvironment();

// read from elements
DataSet<String> names = env.fromElements("Some", "Example", "Strings");

// read from Java collection
List<String> list = new ArrayList<String>();
list.add("Some");
list.add("Example");
list.add("Strings");

DataSet<String> names = env.fromCollection(list);
Data Sources: File-Based

ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

// read text file from local or distributed file system
DataSet<String> localLines =
    env.readTextFile("/path/to/my/textfile");

// read a CSV file with three fields
DataSet<Tuple3<Integer, String, Double>> csvInput =
    env.readCsvFile("/the/CSV/file")
        .types(Integer.class, String.class, Double.class);

// read a CSV file with five fields, taking only two of them
DataSet<Tuple2<String, Double>> csvInput =
    env.readCsvFile("/the/CSV/file")
        .includeFields("10010")
        .types(String.class, Double.class);
Data Sinks

Text
- `writeAsText("/path/to/file")`
- `writeAsFormattedText("/path/to/file", formatFunction)`

CSV
- `writeAsCsv("/path/to/file")`

Return data to the Client
- `Print()`
- `Collect()`
- `Count()`
Data Sinks (lazy)

- Lazily executed when `env.execute()` is called

```java
DataSet<...> result;

// write DataSet to a file on the local file system
result.writeAsText("/path/to/file");

// write DataSet to a file and overwrite the file if it exists
result.writeAsText("/path/to/file", FileSystem.WriteMode.OVERWRITE);

// tuples as lines with pipe as the separator "a|b|c"
result.writeAsCsv("/path/to/file", 
"\n", "|");

// this writes values as strings using a user-defined TextFormatter object
result.writeAsFormattedText("/path/to/file",
    new TextFormatter<Tuple2<Integer, Integer>>(){
        public String format (Tuple2<Integer, Integer> value) {
            return value.f1 + " - " + value.f0;
        }
    });
```
Data Sinks (eager)

- Eagerly executed

```java
DataStream<Tuple2<String, Integer>> result;

// print
result.print();

// count
int numberOfElements = result.count();

// collect
List<Tuple2<String, Integer>> materializedResults = result.collect();
```
Execution Setups
Ways to Run a Flink Program

- **DataSet (Java/Scala/Python)**
- **DataStream (Java/Scala)**

Options:
- Local
- Remote
- Yarn
- Tez
- Embedded

Streaming dataflow runtime
Local Execution

- Starts local Flink cluster
- All processes run in the same JVM
- Behaves just like a regular Cluster
- Very useful for developing and debugging
Embedded Execution

- Runs operators on simple Java collections
- Lower overhead
- Does not use memory management
- Useful for testing and debugging
Remote Execution

- The cluster mode
- Submit a Job remotely
- Monitors the status of the job
YARN Execution

- Multi user scenario
- Resource sharing
- Uses YARN containers to run a Flink cluster
- Very easy to setup Flink
Execution

- Leverages Apache Tez’s runtime
- Built on top of YARN
- Good YARN citizen
- Fast path to elastic deployments
- Slower than Flink
http://dataArtisans.github.io/eit-summerschool-15

Exercises and Hands-on
Closing
tl;dr: What was this about?

- The case for Flink
  - Low latency
  - High throughput
  - Fault-tolerant
  - Easy to use APIs, library ecosystem
  - Growing community

- A stream processor that is great for batch analytics as well
I ♥️ 🐿️, do you?

- Get involved and start a discussion on Flink’s mailing list
  - { user, dev }@flink.apache.org

- Subscribe to news@flink.apache.org
- Follow flink.apache.org/blog and @ApacheFlink on Twitter
flink-forward.org

October 12-13, 2015

Call for papers deadline: August 14, 2015

Discount code: FlinkEITSummerSchool25
Thank you for listening!