**Flink官网学习笔记**

郭元

目录

[Introduction to Apache Flink 3](#_Toc507615524)

[对无界数据的连续处理 3](#_Toc507615525)

[1 数据集的两种类型 3](#_Toc507615526)

[2 两种执行模型 3](#_Toc507615527)

[Flink的特点。 3](#_Toc507615528)

[流式计算模型和有界批处理。 6](#_Toc507615529)

[Flink的构建 7](#_Toc507615530)

[flink的部署 7](#_Toc507615531)

[运行时 7](#_Toc507615532)

[API 7](#_Toc507615533)

[库 7](#_Toc507615534)

[Flink和其他框架 8](#_Toc507615535)

[关键点和下一步计划 8](#_Toc507615536)

[Quickstart 9](#_Toc507615537)

[安装：下载并启动Flink 9](#_Toc507615538)

[以local模式启动Flink 9](#_Toc507615539)

[阅读代码 9](#_Toc507615540)

[运行例子 9](#_Toc507615541)

[下一步 9](#_Toc507615542)

# Introduction to Apache Flink

## 对无界数据的连续处理

在我们介绍Flink的之前，让我们复习一下数据集高层次的类型，我们会在处理数据和执行我们选择执行的模型的时候遇到它。尽管他们经常是在一起的，但是分别介绍他们可以使我们更加清晰的了解他们。

### 1 数据集的两种类型

* 有界数据集：连续追加的无限数据集
* 无界数据集：不能改变的有限数据集

许多传统意义上被认为是有界或者批量的数据实际上是无界数据集。不论数据以序列化的方式存储在HDFS上还是基于日志的系统，例如Kafka，都是如此。

以下列举了一部分无界数据集：

* 用户与移动或者Web应用程序交互产生的数据；
* 物理传感器提供的测量数据；
* 金融市场；
* 机器日志数据

### 2 两种执行模型

* 流计算：只要数据不断的产生就需要连续的执行；
* 批处理：能够在有限的时间内执行完任务，最终释放计算资源。

尽管不一定是最佳的，但可以用任意类型的执行模型处理任意类型的数据集。例如，尽管在窗口化，状态管理和无序数据方面可能存在问题，但批处理执行早已应用于无界数据集。

Flink依赖流式执行模型，直观上适用于处理无界数据集的流式执行模型：流式执行是对连续生成的数据进行连续处理。数据集类型与执行模型类型之间的对齐在准确性和性能方面提供了许多优势。

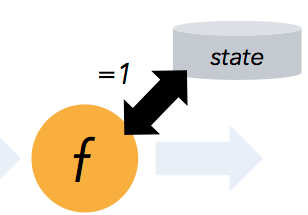
## Flink的特点。

Flink是分布式流式处理的开源的框架：

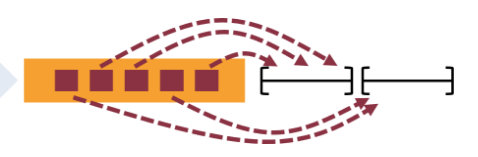
* 结果更准确，即使对于无序或者延迟到达的数据；
* 有状态和容错，可以在保持一次应用程序状态的同时无缝地从故障中恢复；
* 在扩展性方面，能够运行在数千台节点上，具有非常好的吞吐量和延迟特性。

早些时候，我们讨论了将数据集的类型（有界与无界）与执行模型的类型（批量还是流处理）。Flink在流式执行模型方面启用的对状态管理、无序数据的处理、灵活的窗口等功能，其在无界数据集上计算结果的准确性方面是非常重要的。

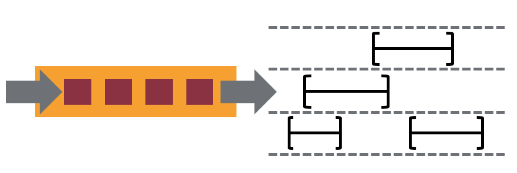
Flink对于有状态的计算保证有且仅有一次计算语义。“有状态”意味着应用程序可以维护一段时间内处理过数据的汇总或者摘要信息，并且Flink的检查点机制保证应用程序失败时有且仅有一次的语义。



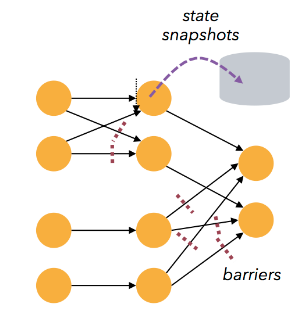
Flink支持使用事件时间语义进行流处理和窗口化。事件时间可以轻松保证事件到达顺序不正确以及事件延迟到达时流处理结果的准确性。



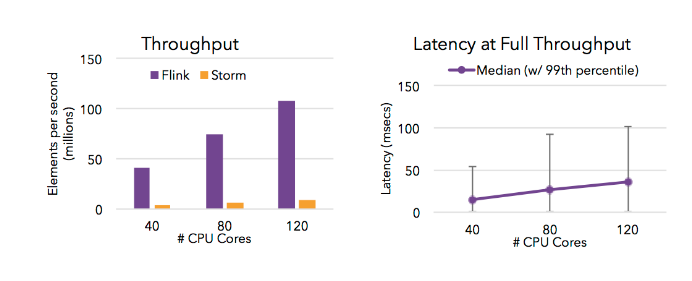
除了数据驱动的窗口计算外，Flink还支持基于时间、计数、会话的窗口计算。窗口计算可以通过灵活的触发条件进行定制，以支持复杂的流处理模型。Flink的窗口化很好的模拟了现实环境中数据创建的情况。



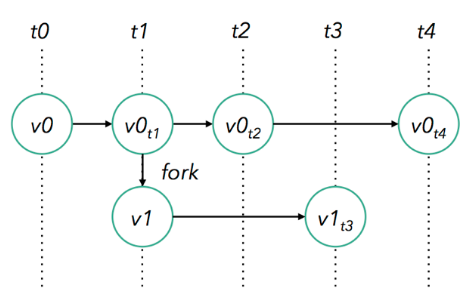
Flink的容错功能非常轻巧，可以让系统保持高吞吐量的同时提供有且仅有一次的保证。当忽略掉可靠性和延迟之间，Flink可以从零数据丢失的故障中恢复。



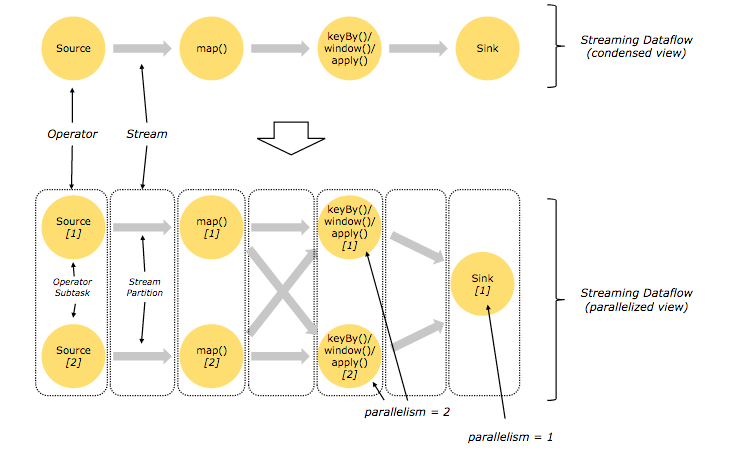
Flink能够实现高吞吐量和低延迟（快速处理大量数据）。下图显示了Apache Flink和Apache Storm在涉及数据shuffle的计数计算的性能对比。



Flink的保存点提供了一种状态版本管理机制，可以更新应用程序或重新处理历史数据，而不会出现丢失状态并且停机时间很短。



Flink可以在数千个节点的大型集群上运行，除了支持伪分布集群模式外，Flink还提供yarn和Mesos的支持。



## 流式计算模型和有界批处理。

如果你已阅读Flink的文档，你可以已经注意到用于处理无界数据集的DataStream API和用于处理有界数据集的DataSet API。

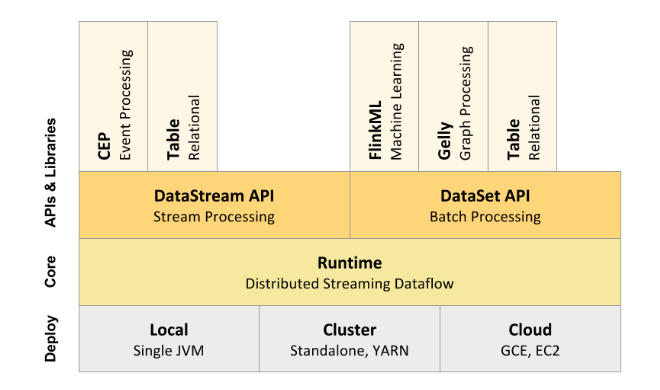
在这片文章的前面，我们介绍了流式执行模型（连续不断的执行处理，基于时间的事件处理）可以作为无界数据集的直观拟合。那么有界数据集如何依赖于流处理。

在Flink中，这种关系是非常自然的，有界数据集可以简单地视为无界数据集的一个特例，因此，我们可以将上面提出的所有相同的流概念应用于有界数据集。

这正式Flink中DataSet API的工作方式。有界数据集在Flink内部作为有限流进行处理，Flink在管理有界数据集合无界数据集只有一些细微差异而已。

因此，可以使用Flink来处理有界数据集合无界数据集，这两个API可以在相同的分布式流式执行引擎上运行---这是一个简单而强大的体系结构。

## Flink的构建



### flink的部署

Flink可以运行在云环境中，或者本地环境中，或者伪分布集群中，或者由Yarn或Mesos管理集群上。

### 运行时

Flink的核心是一个分布式流处理引擎，这意味着数据一次处理而不是一些列批处理，这与其他流处理框架是一个重要的区别，其导致Flink许多弹性话和性能特点成为可能。

### API

* Flink的DataStream API适用于数据流转化（例如过滤，更新状态，定义窗口，聚合）。
* Flink的DataSet API适用于数据集的转化（例如过滤，映射，连接，分组）。
* Tabel API用于关系流和批处理的SQL式表达语言，可以轻松嵌入到Flink的DataSet和DataStream API（Java和Scala）中。
* Streaming SQL使SQL查询能够在流处理和批处理上，该语法基于Apache CalciteTM

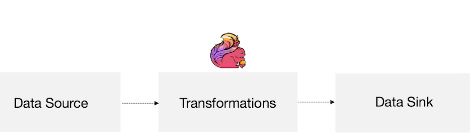
### 库

Flink还包括用于复杂时间处理，机器学习，图计算和Apache Storm兼容的专用库。

## Flink和其他框架

在最基本的层面上，Flink程序由一下部分组成：

* Data source:数据源
* Transformations:数据处理部分
* Data sink：数据处理结束以后Flink发送结果的地方



一个完善的生态系统对于数据的接入和输出的支持是必要的，Flink支持多种数据源接入和输出的连接器。

## 关键点和下一步计划

略。

# Quickstart

## 安装：下载并启动Flink

Flink可在Linux，Mac OS X和Windows上运行。 为了能够运行Flink，唯一的要求是安装Java 7.x（或更高版本）。 Windows用户请查看Windows上的Flink指南，其中介绍了如何在Windows上运行Flink以进行本地设置。

您可以通过发出以下命令来检查Java的正确安装：

|  |
| --- |
| java version "1.8.0\_111"  Java**(**TM**)** SE Runtime Environment **(**build 1.8.0\_111-b14**)**  Java HotSpot**(**TM**)** 64-Bit Server VM **(**build 25.111-b14, mixed mode**)** |

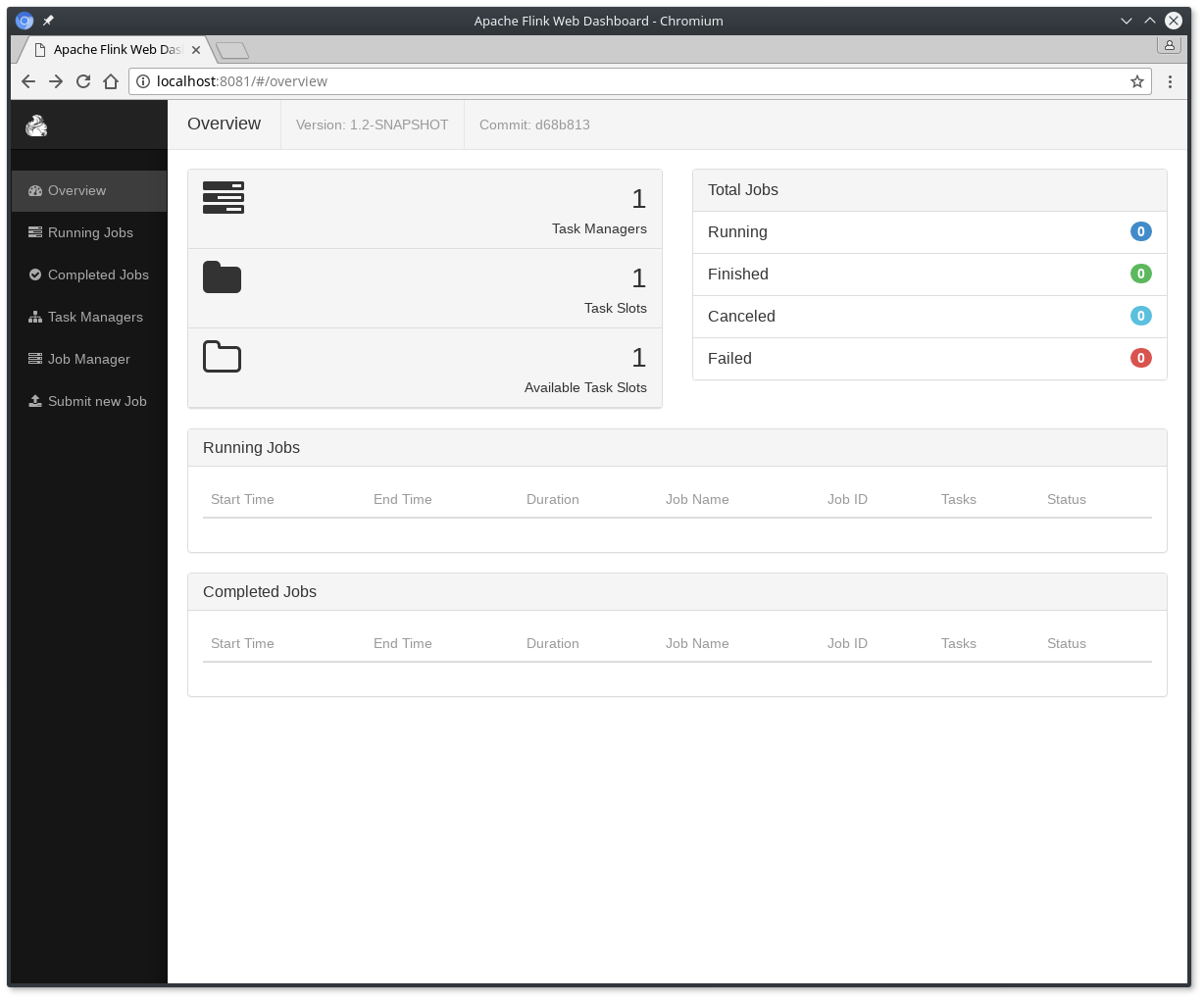
1. 从下载页面下载二进制文件。 你可以选择你喜欢的任何Hadoop / Scala组合。 如果您打算只使用本地文件系统，则任何Hadoop版本都可以正常工作。
2. 转到下载目录。
3. 解压下载的二进制文件。

|  |
| --- |
| $ cd ~/Downloads *# Go to download directory*  $ tar xzf flink-**\***.tgz *# Unpack the downloaded archive*  $ cd flink-1.4.1 |

### 以local模式启动Flink

|  |
| --- |
| $ ./bin/start-local.sh *# Start Flink* |

在http：// localhost：8081检查JobManager的Web前端，确保一切正常并运行。 Web前端应报告一个可用的TaskManager实例。



您还可以通过检查日志目录中的日志文件来验证系统是否正在运行：

|  |
| --- |
| $ tail log/flink-**\***-jobmanager-**\***.log  INFO ... - Starting JobManager  INFO ... - Starting JobManager web frontend  INFO ... - Web frontend listening at 127.0.0.1:8081  INFO ... - Registered TaskManager at 127.0.0.1 **(**akka://flink/user/taskmanager**)** |

## 阅读代码

您可以在GitHub上的scala和java中找到此SocketWindowWordCount示例的完整源代码。

|  |
| --- |
| **object** **SocketWindowWordCount** **{**  **def** main**(**args**:** **Array[String])** **:** **Unit** **=** **{**  *// the port to connect to*  **val** port**:** **Int** **=** **try** **{**  **ParameterTool.**fromArgs**(**args**).**getInt**(**"port"**)**  **}** **catch** **{**  **case** e**:** **Exception** **=>** **{**  **System.**err**.**println**(**"No port specified. Please run 'SocketWindowWordCount --port <port>'"**)**  **return**  **}**  **}**  *// get the execution environment*  **val** env**:** **StreamExecutionEnvironment** **=** **StreamExecutionEnvironment.**getExecutionEnvironment  *// get input data by connecting to the socket*  **val** text **=** env**.**socketTextStream**(**"localhost"**,** port**,** '\n'**)**  *// parse the data, group it, window it, and aggregate the counts*  **val** windowCounts **=** text  **.**flatMap **{** w **=>** w**.**split**(**"\\s"**)** **}**  **.**map **{** w **=>** **WordWithCount(**w**,** 1**)** **}**  **.**keyBy**(**"word"**)**  **.**timeWindow**(Time.**seconds**(**5**),** **Time.**seconds**(**1**))**  **.**sum**(**"count"**)**  *// print the results with a single thread, rather than in parallel*  windowCounts**.**print**().**setParallelism**(**1**)**  env**.**execute**(**"Socket Window WordCount"**)**  **}**  *// Data type for words with count*  **case** **class** **WordWithCount(**word**:** **String,** count**:** **Long)**  **}** |

## 运行例子

现在，我们要运行这个Flink应用程序。 它将从套接字中读取文本，并且每5秒打印一次在前5秒内每个不同单词的出现次数。

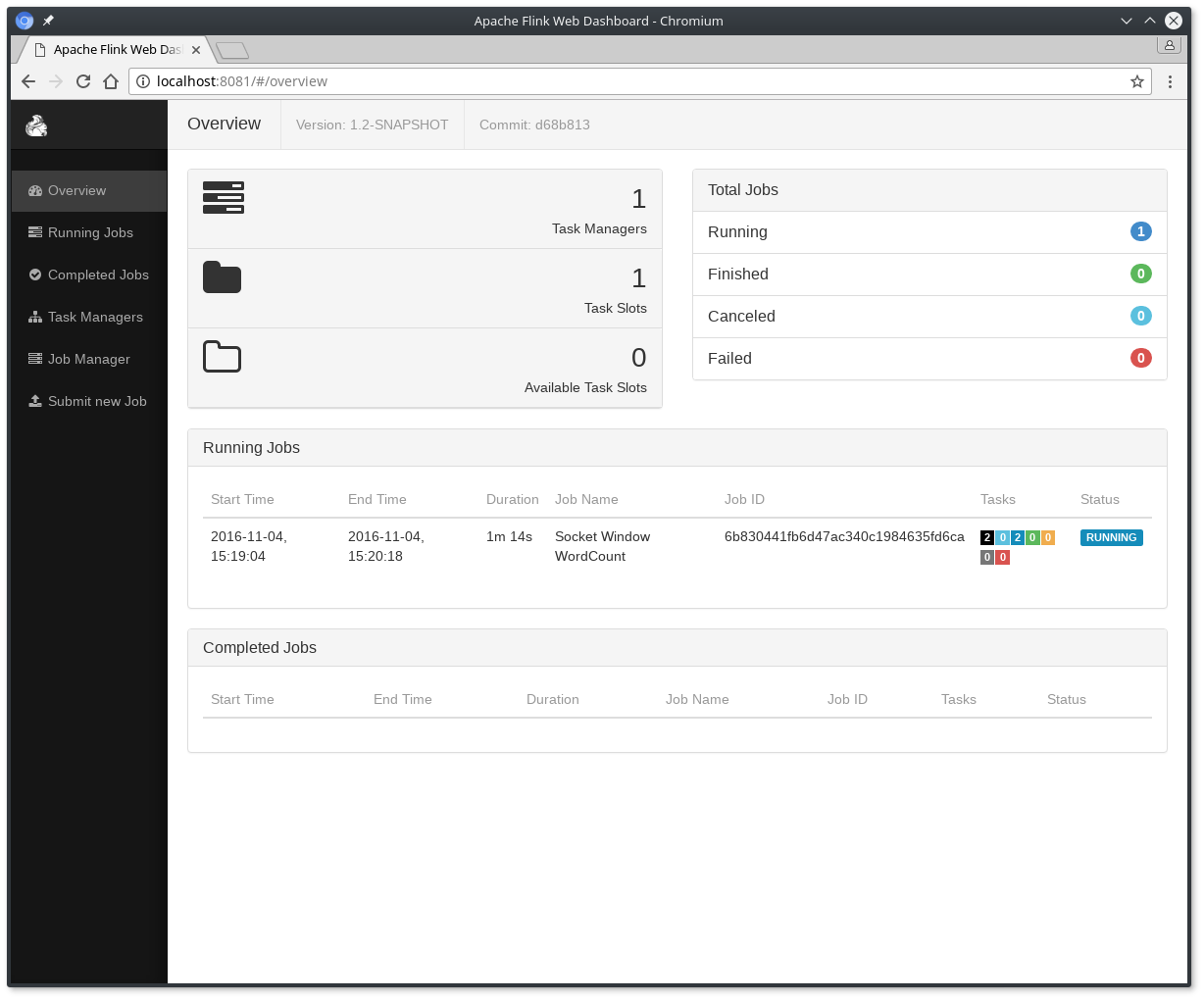
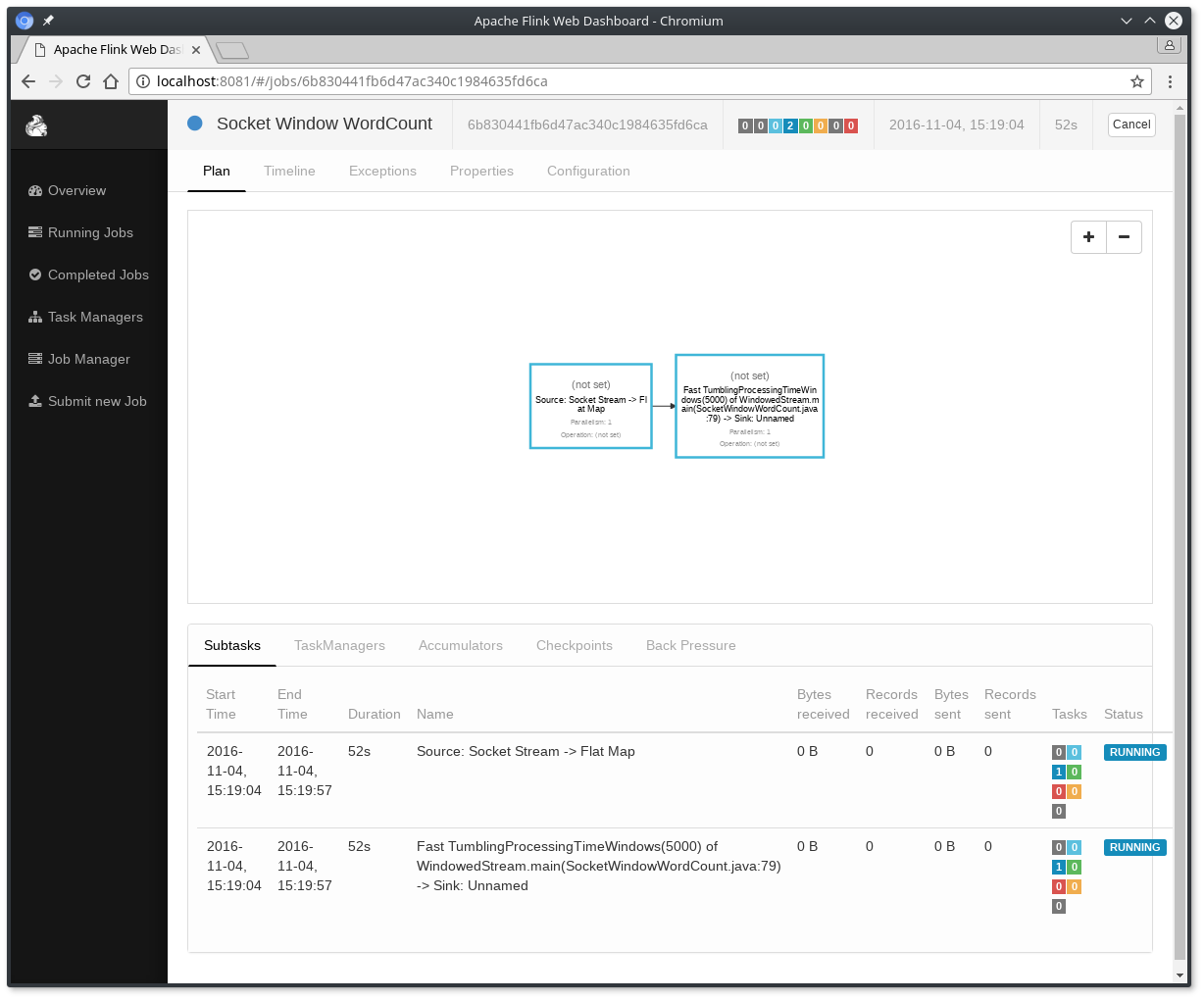
* 首先，我们使用netcat来启动本地服务器：

|  |
| --- |
| $ nc -l 9000 |

* 提交Flink程序：

|  |
| --- |
| $ ./bin/flink run examples/streaming/SocketWindowWordCount.jar --port 9000  Cluster configuration: Standalone cluster with JobManager at /127.0.0.1:6123  Using address 127.0.0.1:6123 to connect to JobManager.  JobManager web interface address http://127.0.0.1:8081  Starting execution of program  Submitting job with JobID: 574a10c8debda3dccd0c78a3bde55e1b. Waiting **for** job completion.  Connected to JobManager at Actor[akka.tcp://flink@127.0.0.1:6123/user/jobmanager#297388688]  11/04/2016 14:04:50 Job execution switched to status RUNNING.  11/04/2016 14:04:50 Source: Socket Stream -> Flat Map**(**1/1**)** switched to SCHEDULED  11/04/2016 14:04:50 Source: Socket Stream -> Flat Map**(**1/1**)** switched to DEPLOYING  11/04/2016 14:04:50 Fast TumblingProcessingTimeWindows**(**5000**)** of WindowedStream.main**(**SocketWindowWordCount.java:79**)** -> Sink: Unnamed**(**1/1**)** switched to SCHEDULED  11/04/2016 14:04:51 Fast TumblingProcessingTimeWindows**(**5000**)** of WindowedStream.main**(**SocketWindowWordCount.java:79**)** -> Sink: Unnamed**(**1/1**)** switched to DEPLOYING  11/04/2016 14:04:51 Fast TumblingProcessingTimeWindows**(**5000**)** of WindowedStream.main**(**SocketWindowWordCount.java:79**)** -> Sink: Unnamed**(**1/1**)** switched to RUNNING  11/04/2016 14:04:51 Source: Socket Stream -> Flat Map**(**1/1**)** switched to RUNNING |

程序连接到套接字并等待输入。 您可以检查Web界面以验证作业是否按预期运行：

* 单词在5秒（处理时间，翻滚窗口）的时间窗口中计数并打印到标准输出。 监视TaskManager的输出文件并在nc中写入一些文本（输入在命中后逐行发送到Flink）：

|  |
| --- |
| $ nc -l 9000  lorem ipsum  ipsum ipsum ipsum  bye |

.out文件将在每个时间窗口结束时打印计数，只要字是浮动的，例如：

|  |
| --- |
| $ tail -f log/flink-**\***-taskmanager-**\***.out  lorem : 1  bye : 1  ipsum : 4 |

完成后要停止Flink，请输入：

|  |
| --- |
| $ ./bin/stop-local.sh |

## 下一步

略