Федеральное государственное автономное образовательное учреждение высшего образования
«Самарский национальный исследовательский университет имени академика С.П. Королёва»

Факультет информатики

Кафедра технической кибернетики

Лабораторная работа №1
по дисциплине
«Большие данные»

Введение в Apache Spark

Введение

Apache Spark — программный каркас с открытым исходным кодом для реализации распределённой обработки неструктурированных и слабоструктурированных данных, входящий в экосистему проектов Hadoop. В отличие от классического обработчика из ядра Hadoop, реализующего концепцию MapReduce с дисковым вводом и выводом, Spark использует специализируется на обработке в оперативной памяти, благодаря чему позволяет получать значительный выигрыш в скорости работы для некоторых классов задач. В частности, возможность многократного доступа к загруженным в память пользовательским данным делает библиотеку привлекательной для алгоритмов машинного обучения.

Главной абстракцией Spark фреймворка является распределённая коллекция элементов Resilient Distributed Dataset (RDD). К RDD можно применить трансформации (transformation) и действия (action). В первом случае в качестве результата возвращается ссылка на новый RDD, а во втором, вычисленное значение цепочки трансформаций.

В папке с заданием содержатся следующие наборы данных:

- книга (warandpeace.txt),
- https://databank.illinois.edu/datasets/IDB-9610843 данные о такси Нью-Йорка за 2013 год (nyctaxi.csv),
- https://www.kaggle.com/benhamner/sf-bay-area-bike-share данные велопарковок Сан-Франциско (trips.csv, stations.csv).

Цель работы

- изучить операции загрузки и выгрузки данных в HDFS,
- ознакомиться с базовыми операциями Apache Spark в spark-shell,
- создать проект по обработке данных в IDE,
- отладить анализ данных велопарковок на локальном компьютере,
- запустить анализ данных велопарковок на сервере.

Настройка окружения

Скачайте и запустите виртуальную машину с установленными Hadoop и Spark. Например:

- MapR https://mapr.com/try-mapr/sandbox/,
- Hortonworks https://www.cloudera.com/downloads/hortonworks-sandbox.html .

Либо, следуя инструкциями в приложении В, запустите Spark кластер в облаке Microsoft Azure.

MapR-FS — API совместимая с HDFS реализация распределённой файловой системы (РФС) от MapR.

Подключитесь к машине по ssh, используя ip или имя узла кластера. Узнать ip адрес виртуальной машины можно выполнив команду ifconfig. В облаке вашей машине присваивается имя, которое вы можете найти на главной странице HDInsight кластера в вкладке Overview.

Работа в консоли spark-shell

В первой части задания вы работаете с **3** файловыми системами:

- локальной файловой системой, в которой находятся файлы с заданиями,
- файловой системой Linux узла кластера,
- распределённой файловой системой (краткое описание приведено в приложении А).

Основные операции взаимодействия с распределённой файловой системой

Для импорта/экспорта данных в РФС используйте команды hadoop fs –put и hadoop fs –get:

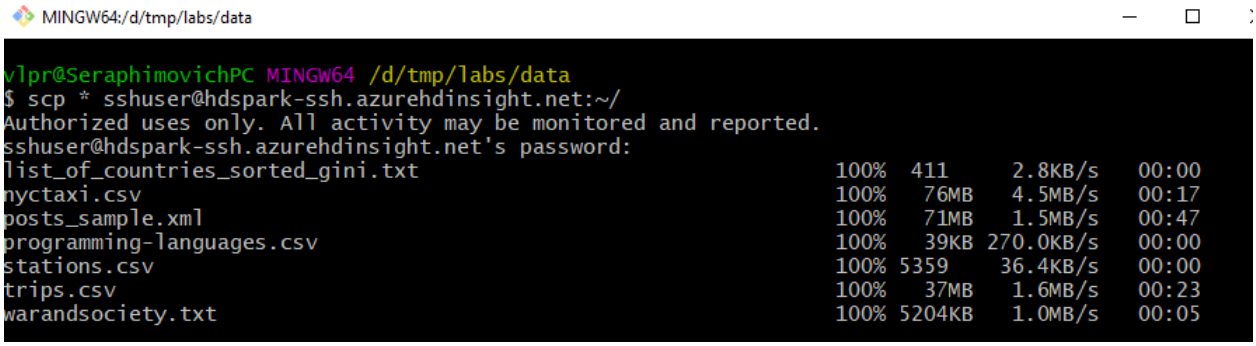
```
$ hadoop fs -put путь-в-локальной-системе путь-в-hdfs
```

```
$ hadoop fs -get путь-в-hdfs путь-в-локальной-системе
```

Список остальных команд для взаимодействия с РФС вы можете посмотреть, выполнив hadoop fs без дополнительных ключей. Мы рассмотрим примеры работы с наиболее полезными командами далее.

Переместите необходимые для работы наборы данных в РФС. Для этого сначала скопируйте их в файловую систему узла кластера с помощью команды scp, утилиты WinSCP или плагина NetBox Far менеджера.

```
$ scp * mapr@192.168.81.129:~/
mapr@192.168.81.129's password:
list_of_countries_sorted_gini.txt                100%  411    14.7KB/s   00:00
nyctaxi.csv                                      100%   76MB  48.2MB/s   00:01
posts_sample.xml                                 100%   71MB  49.4MB/s   00:01
programming-languages.csv                        100%   39KB   9.2MB/s   00:00
stations.csv                                     100% 5359     1.4MB/s   00:00
trips.csv                                        100%   37MB  38.8MB/s   00:00
warandsociety.txt                                100% 5204KB  55.1MB/s   00:00
```



Затем на удалённой машине, находясь в директории с перемещёнными файлами, используйте команду

```
$ hadoop fs -put * .
```

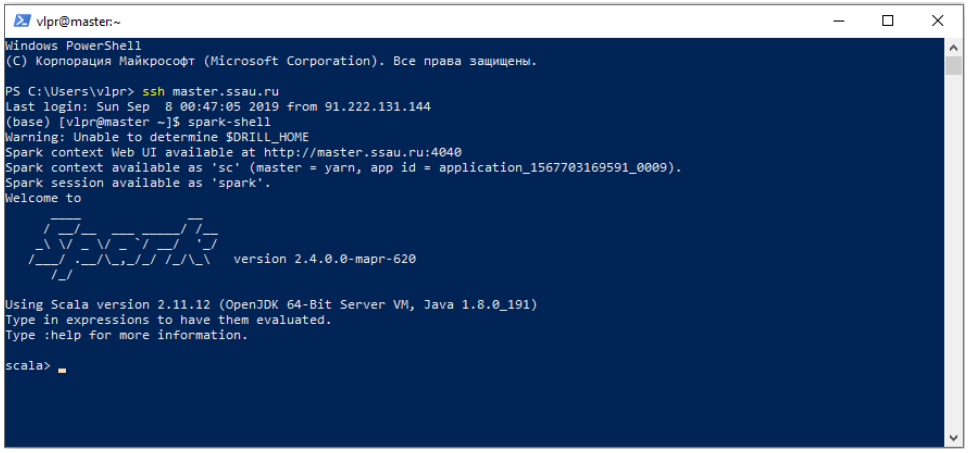
Проверьте, переместились ли файлы.

```
$ hadoop fs -ls
```

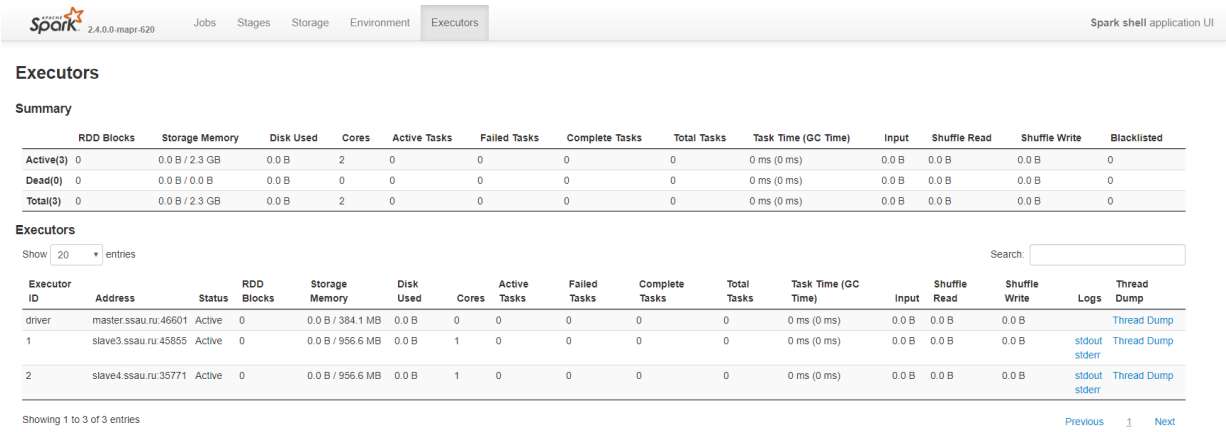
Следует обратить внимание на то, что в команде не указывалась директория, то есть использовалась директория по умолчанию.

Попробуйте другие команды, например -mkdir, -cat, -df, -du. После того как вы освоитесь с перемещением данных в РФС, запустите spark-shell.

```
$ /opt/mapr/spark/spark-2.3.1/bin/spark-shell
```



Веб-страница запущенной сессии spark-shell.



Примечание. Попытка привязки веб интерфейса начинается с номера порта 4040 и продолжаются с увеличением на единицу пока не будет найден свободный порт.

Зайдите на веб-страницу менеджера ресурсов YARN. Ваше приложение должно присутствовать в списке в статусе RUNNING. Spark может быть запущен в нескольких режимах: локальном, standalone (собственный менеджер ресурсов), yarn и mesos (внешние менеджеры ресурсов).

В Spark History Server вы можете посмотреть информацию о выполненных Spark задачах.

Таблица 1. Порты сервисов в виртуальной машине MapR

Сервис	Порт
YARN	8090
Spark History Server	18080
MapR Control System	8443

Создание Resilient Distributed Dataset

Создайте RDD для текстового файла warandpeace.txt. Для подробного списка операций считывания файлов обращайтесь к документации класса SparkContext https://spark.apache.org/docs/2.4.0/api/scala/index.html#org.apache.spark.SparkContext.

Примечание. При наборе команд используйте TAB --- функцию автодополнения.

```
val warandpeace = sc.textFile("warandpeace.txt")
```

В данной команде указывается относительный путь, который начинается с вашей папки в РФС.

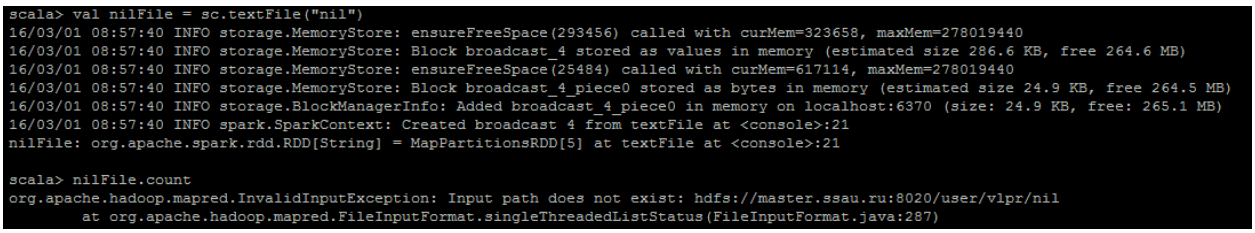
Выведите количество строк файла.

```
warandpeace.count
```

Примечание. При отсутствии у функции аргументов, в scala скобки можно опускать.

Попробуйте считать несуществующий файл, например *nil*, а затем вывести количество его строк на экран

```
val nilFile = sc.textFile("nil")
nilFile.count
```



Заметьте, что первая команда выполняется успешно, а вторая выводит сообщение, что такого файла нет. Это происходит потому, что выполнение обработки в Spark является ленивым и не запускается, до встречи команды действия(action). count - первая команда действия, с которой вы познакомились.

Считайте первые 10 строк файла warandpeace.txt.

```
warandpeace.take(10)
```

Эта команда не требует считывания и передачи на главный узел всех данных RDD.

Узнайте на сколько частей разделились данные в кластере.

```
warandpeace.partitions
```

Если используется определённый метод распределения вы можете получить данные о нём командой *partitioner*. Начиная с версии 1.6.0 доступна команда *warAndPeaceFile.getNumPartitions* для получения информации о количестве разделов.

Создайте распределённую коллекцию из нескольких элементов и для каждого элемента верните ip адрес, на котором он расположен:

```
sc.parallelize(Array(1,2,3)).map(x =>
java.net.InetAddress.getLocalHost).collect
```

Обработка текста

Найдите строки, в которых содержится слово "война".

```
val linesWithWar = warAndPeaceFile.filter(x =>
x.contains("война"))
```

Примечание. Аргументом filter является лямбда функция - функция без имени. До обозначения ***=>*** в скобках через запятую следуют переменные аргументов функции, затем следует команда языка Scala. При использовании фигурных скобок язык позволяет описывать лямбда функции с цепочкой команд в теле, аналогично именованным функциям.

Запросите первую строку. Строкой в данном файле является целый абзац, так как только по завершению абзаца содержится символ переноса строки.

```
linesWithWar.first
```

Данные могут быть перемещены в кэш. Этот приём очень полезен при повторном обращении к данным, для запросов "горячих" данных или запуска итеративных алгоритмов.

Перед подсчётом количества элементов вызовите команду кэширования cache(). Трансформации не будут обработаны, пока не будет запущена одна из команд - действий.

```
linesWithWar.cache()
linesWithWar.count()
linesWithWar.count()
```

Можете воспользовать следующим блоком кода для замера времени выполнения команды.

```
def time[R](block: => R): R = {
    val t0 = System.nanoTime()
    val result = block    // call-by-name
    val t1 = System.nanoTime()
    println("Elapsed time: " + (t1 - t0) + "ns")
    result
}
```

```
count at <console>:26, took 0,066794 s

count at <console>:26, took 0,014998 s
```

```
scala> val linesWithWar = warandpeace.filter(x => x contains "война")
linesWithWar: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[3] at filter at <console>:25

scala> time { linesWithWar.count }
Elapsed time: 214709872ns
res22: Long = 54

scala> time { linesWithWar.count }
Elapsed time: 136522537ns
res23: Long = 54

scala> linesWithWar.cache()
res24: linesWithWar.type = MapPartitionsRDD[3] at filter at <console>:25

scala> time { linesWithWar.count }
Elapsed time: 152316423ns
res25: Long = 54

scala> time { linesWithWar.count }
Elapsed time: 55745366ns
res26: Long = 54

scala> time { linesWithWar.count }
Elapsed time: 57074725ns
res27: Long = 54
```

При выполнении команды count второй раз вы должны заметить небольшое ускорение. Кэширование небольших файлов не даёт большого преимущества, однако для огромных файлов, распределённых по сотням или тысячам узлов, разница во времени выполнения может быть существенной. Вторая команда linesWithWar.count() выполняется над результатом от предшествующих команде cache трансформаций и на больших объёмах данных будет ускорять выполнение последующих команд.

Найдите гистограмму слов:

```
val wordCounts = linesWithWar.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey((a, b) => a + b)
```

Spark существенно упростил реализацию многих задач, ранее решаемых с использованием MapReduce. Эта однострочная программа --- WordCount --- является наиболее популярным примером задачи, эффективно распараллеливаемой в Hadoop кластере. Её реализация в MapReduce занимается около 130 строк кода.

Сохраните результаты в файл, а затем, найдите данные в HDFS и выведите данные в linux консоли с помощью команды hadoop fs -cat warandpeace_histogram.txt/* (здесь используется относительный путь).

```
wordCounts.saveAsTextFile("warandpeace_histogram.txt")
```

```
$ hadoop fs -cat warandpeace_histogram.txt/*
```

Упражнение. Улучшите процедуру, убирая из слов лишние символы и трансформируя все слова в нижний регистр. Используйте регулярные выражения. Например, по регулярному выражению *"\\w*".r* следующий код

```
"\\w*".r.findAllIn("a b c").toArray.foreach(println)
```

выведет на каждой строке по букве. Кроме Scala консоли для тестирования регулярных выражений вы можете использовать сайты:

- https://regex101.com/,
- https://www.debuggex.com/.

Операции с множествами

Инициализируйте два множества

```
val a = sc.parallelize(Array(1,2,3,4))
val b = sc.parallelize(Array(3,4,6,7))
```

Найдите объединение a и b и соберите данные на главный узел с помощью функции collect.

```
a.union(b).collect
```

Обратите внимание, что общие элементы дублируются, поэтому результат не является классическим множеством на самом деле. Такое поведение делает

это операцию очень дешёвой, так как обновляется только информация о местонахождении данных для данного RDD. Уберите дубликаты с помощью distinct.

```
a.union(b).distinct().collect
```

Найдите пересечение множеств.

```
a.intersection(b).collect
```

Найдите разность множеств.

```
a.subtract(b).collect
```

Примечание. При запуске collect на центральный узел - *driver* передаются все данные из распределённого RDD. При работе с большим объемом данных выполнение данной команды может заполнить всю оперативную память *driver* узла.

Упражнение. Найдите в исходном коде Spark определение функции distinct. Объясните почему реализация этой операции действительно убирает дубликаты.

Общие переменные

В Apache Spark общими переменными являются широковещательные (broadcast) переменные и аккумулирующие (accumulator) переменные.

Широковещательные переменные

Общие переменные удобны если вы обращаетесь к небольшому объёму данных на всех узлах. Например, это могут быть параметры алгоритмов обработки, небольшие матрицы.

В консоли, с которой вы работали в предыдущем разделе, создайте широковещательную переменную. Наберите:

```
val broadcastVar = sc.broadcast(Array(1,2,3))
```

Для получения значения обратитесь к полю value:

```
broadcastVar.value
```

Аккумулирующие переменные

Аккумулирующие переменные являются объектами, которые могут быть изменены только ассоциативной операцией добавления. Они используются для эффективной реализации счётчиков и суммарных значений. Вы можете также использовать свой тип, над котором определена ассоциативная операция при необходимости.

Особенностью использования переменной является возможность доступа к значению только на узле в driver процессе.

Потренируйтесь в создании аккумулирующих переменных:

```
val accum = sc.longAccumulator
```

Следующим шагом запустите параллельную обработку массива и в каждом параллельном задании добавьте к аккумулирующей переменной значение элемента массива:

```
sc.parallelize(Array(1,2,3,4)).foreach(x =>
accum.add(x))
```

Для получения текущего значения вызовите команду:

```
accum.value
```

Результатом должно быть число 10.

Пары ключ-значение

Создайте пару ключ-значение из двух букв:

```
val pair = ('a', 'b')
```

Для доступа к первому значению обратитесь к полю _1:

```
pair._1
```

Для доступа к второму значению к полю _2:

```
pair._2
```

Если распределённая коллекция состоит из пар, то они трактуются как для ключ-значение и для таких коллекций доступны дополнительные операции.

Наиболее распространённые, это: группировка по ключу, агрегирование значений с одинаковыми ключами, объединение двух коллекций по ключу.

Вы можете выйти из консоли нажатием сочетания клавиш CTRL+D.

К текущему моменту вы познакомились со следующими командами действий: count, first, take, saveAsTextFile, collect, foreach. Полный список команд действий вы можете найти в документации соответствующей версии Spark http://spark.apache.org/docs/latest/rdd-programming-guide.html#actions.

Десять наиболее популярных номеров такси

Проанализируем данные о поездках такси в Нью-Йорке и найдём 10 номеров такси, которые совершили наибольшее количество поездок.

В первую очередь будет необходимо загрузить данные в MapR-FS. Создайте новую папку в MapR-FS:

Создайте RDD на основе загруженных данных nyctaxi.csv:

```
val taxi = sc.textFile("nyctaxi.csv")
```

Выведите первые 5 строк из данной таблицы:

```
taxi.take(5).foreach(println)
```

Обратите внимание, что первая строка является заголовком. Её как правило нужно будет отфильтровать. Одним из эффективных способов является следующий:

```
taxi.mapPartitionsWithIndex{(idx,iter)=> if (idx==0)
iter.drop(1) else iter }
```

Примечание. Для анализа структурированных табличных данных рассматривайте в качестве альтернативы использование SQL API и DataSet API.

Для разбора значений потребуется создать RDD, где каждая строка разбита на массив подстрок. Используйте запятую в качестве разделителя. Наберите:

```
val taxiParse = taxi.map(line=>line.split(","))
```

Теперь преобразуем массив строк в массив пар ключ-значение, где ключом будет служить номер такси (6 колонка), а значением единица.

```
val taxiMedKey = taxiParse.map(row => (row(6), 1))
```

Следом мы можем найти количество поездок каждого номера такси:

```
val taxiMedCounts = taxiMedKey.reduceByKey((v1, v2) =>
v1+v2)
```

Выведем полученные результаты в отсортированном виде:

```
taxiMedCounts.map(_.swap).top(10).map(_.swap).foreach(p
rintln)
```

Примечание. Нотация _.swap является объявлением анонимной функции от одного аргумента, аналог записи x => x.swap.

Являются ли обе map операции распределёнными? Найдите в документации Spark в классах RDD или PairRDDFunctions метод top.

Вы также можете сгруппировать все описанные выше трансформации, преобразующие исходные данные в одну цепочку:

```
val taxiCounts =
taxi.map(line=>line.split(",")).map(row=>(row(6),1)).re
duceByKey(_ + _)
```

Примечание. Нотация _ + _ является объявлением анонимной функции от двух аргументов, аналог более многословной записи (a,b) => a + b.

Попробуйте найти общее количество номеров такси несколько раз, предварительно объявив RDD taxiCounts как сохраняемую в кэше:

```
taxiCounts.cache()
```

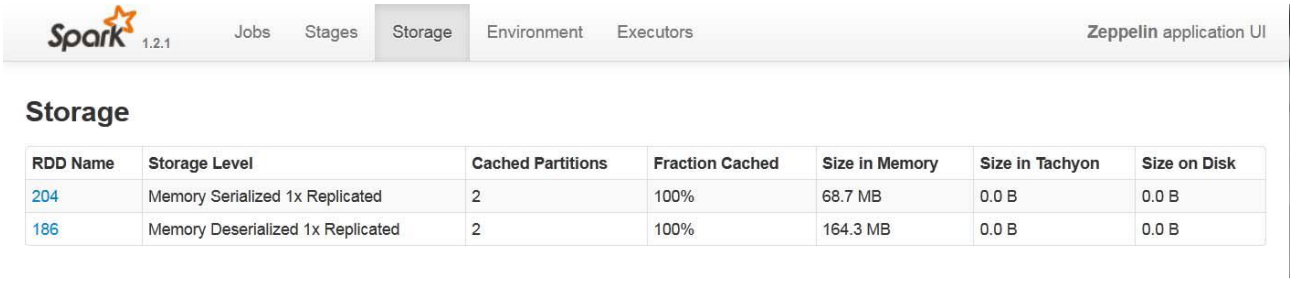
Сравните время, которое трансформации выполняются первый раз и второй. Чем больше данные, тем существеннее разница.

```
taxiCounts.count()
taxiCounts.count()
```

Настройка способа хранения RDD

В данной части будет рассмотрена настройка способов хранения RDD. Вы сравните различные способы хранения, включая: хранение в сериализованном виде, в исходном, с репликацией.

```
trips.persist(StorageLevel.MEMORY_ONLY)
```



Другими способами хранения являются:

- MEMORY_AND_DISK,
- MEMORY_AND_DISK_SER,
- DISK_ONLY,
- MEMORY_ONLY_2,
- MEMORY_AND_DISK_2,
- OFF_HEAP.

Подробнее о способах хранения вы можете узнать по адресу http://spark.apache.org/docs/latest/programming-guide.html#rdd-persistence

Создание проекта на локальном компьютере

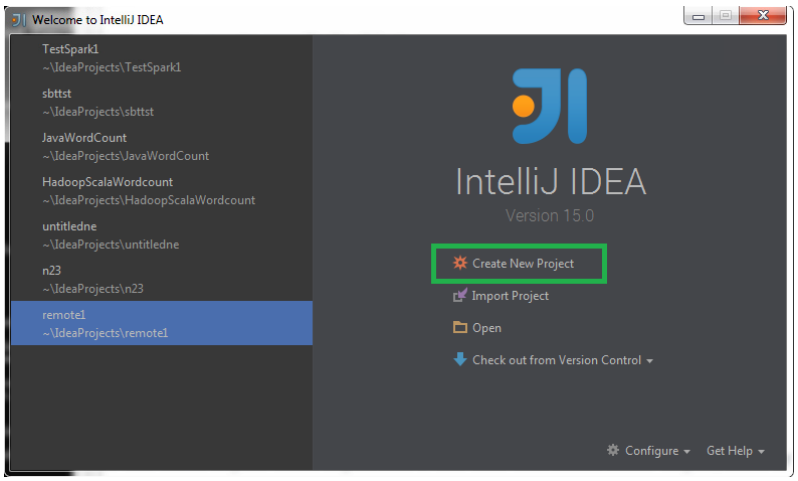
Разработка программы на Spark может быть выполнена на нескольких языках: Python, R, Scala, Java. В данном руководстве рассмотрим разработку на последних двух, так как они имеют самую полную поддержку API обработки данных.

Разработка приложения может производиться в любом текстовом редакторе, затем быть собрана системой сборки в отдельное приложение и запущена на Spark кластере с помощью консольной команды spark-submit.

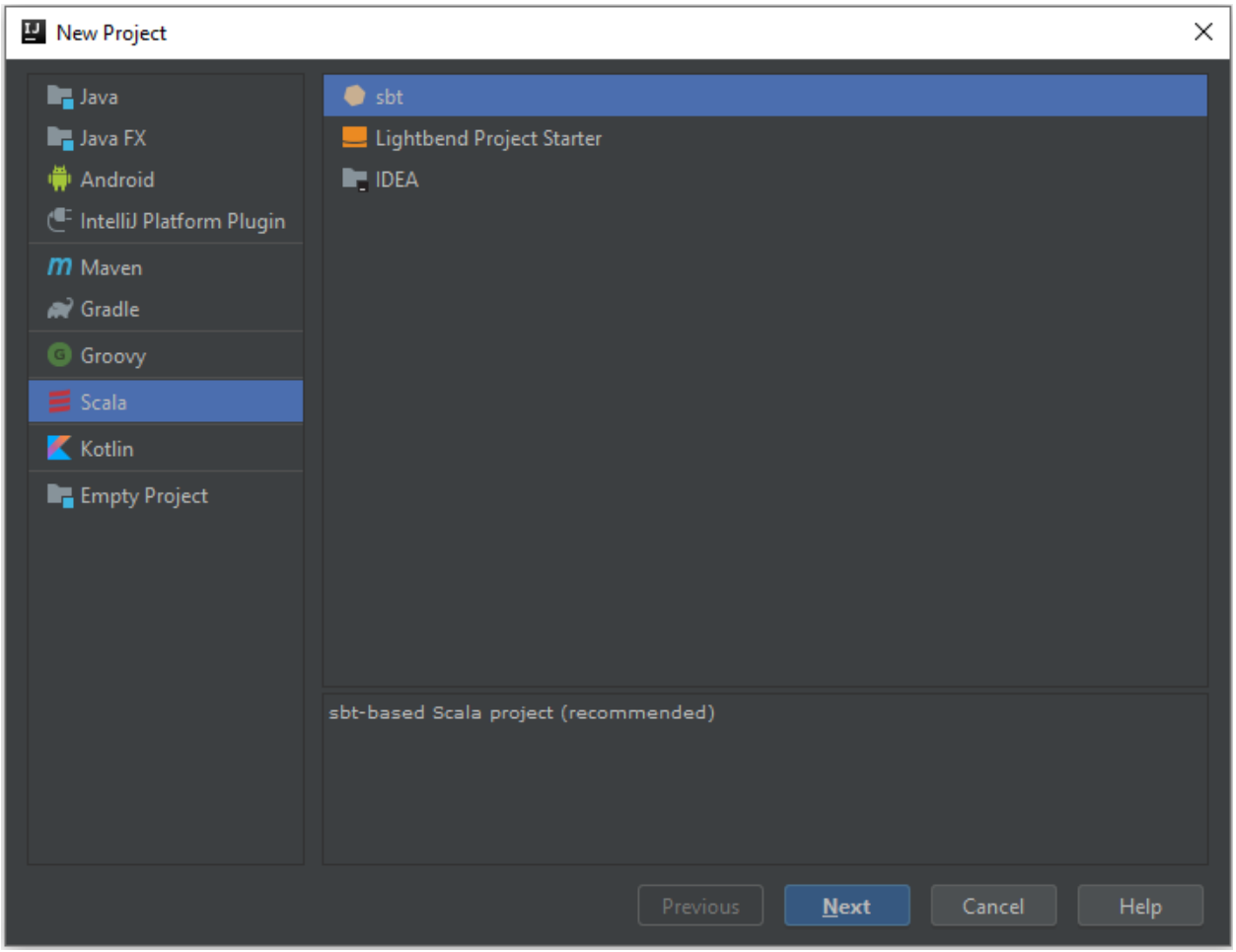
В данной лабораторной работе мы будем использовать IntelliJ IDEA. IDE предоставляет набор возможностей для упрощения разработки: автодополнение, индексация проекта, статическая проверка кода, подсветка синтаксиса, интеграция с системами контроля версий и системами сборки.

Для работы необходима установленная последняя версия IDE IntelliJ IDEA Community Edition. Данная среда разработки доступа для скачивания по адресу https://www.jetbrains.com/idea/.

Для создания проекта в IntelliJ IDEA запустите среду разработки, выберите Create New Project.

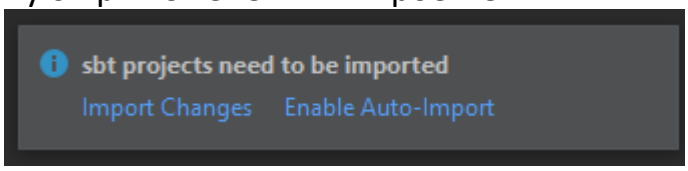


Для создания Scala + SBT проекта выберите слева в меню Scala и затем SBT.



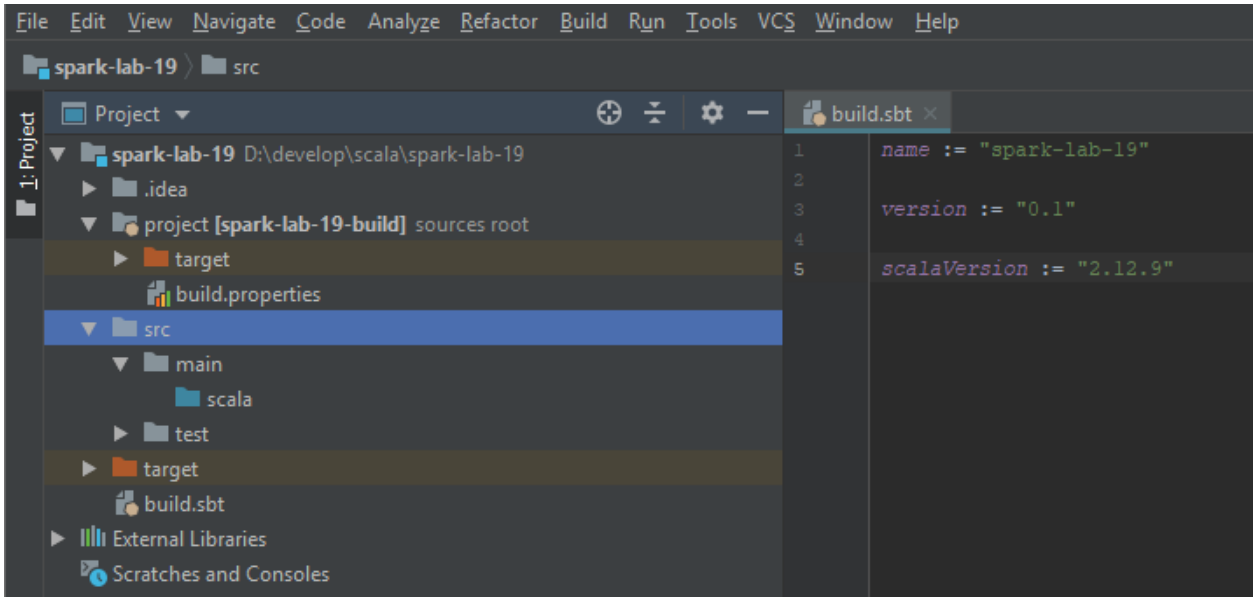
Далее укажите имя проекта, версию Java, версию SBT и версию Scala компилятора. Для разработки на Spark 2.4.0 рекомендуется выбрать версию Scala 2.12.9.

Примечание. Установите флаг Use auto-import для того, чтобы не обновлять зависимости вручную при изменениях в проекте.



После нажатия Finish откроется главное окно среды разработки.

Подождите, когда SBT скачает зависимости.

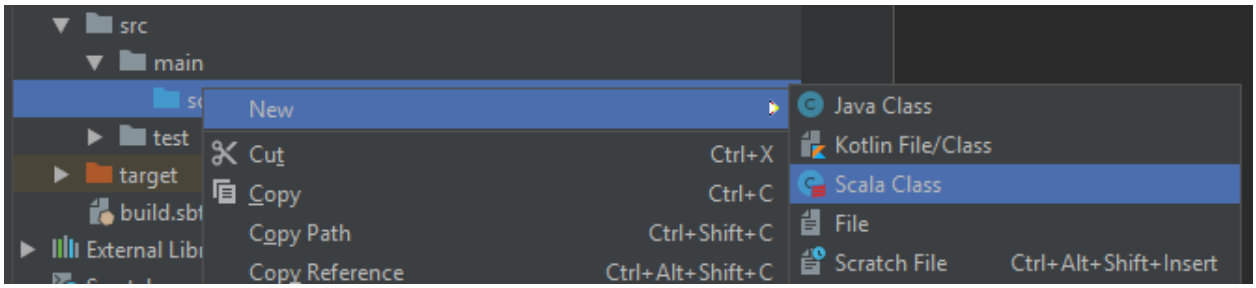


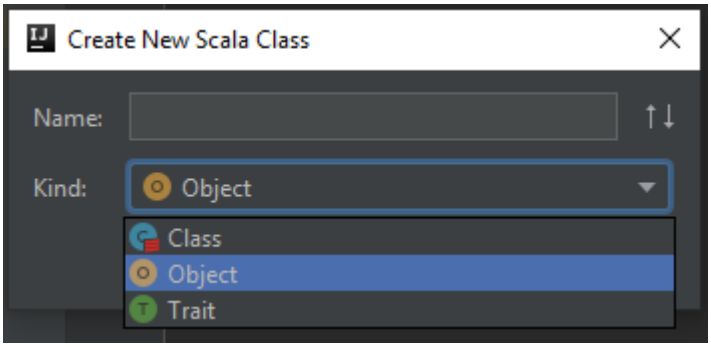
В дереве проекта должен появиться файл build.sbt, являющийся основным файлом для настройки сборки и указания зависимостей проекта SBT. В файле на момент создания указаны: имя проекта, версия проекта, версия языка Scala.

Примечание. Появление предупреждений о конфликте имён в SBT 0.13.8 является известной ошибкой https://github.com/sbt/sbt/issues/1933. Одно из решений — использование более ранней версии или скрытие всех предупреждений установкой степени логирования logLevel := Level.Error.

Код Scala помещается в папку src/main/scala или src/main/scala-2.12.

Создайте в папке scala объект Main с методом main. Данный метод будет точкой входа в программу.

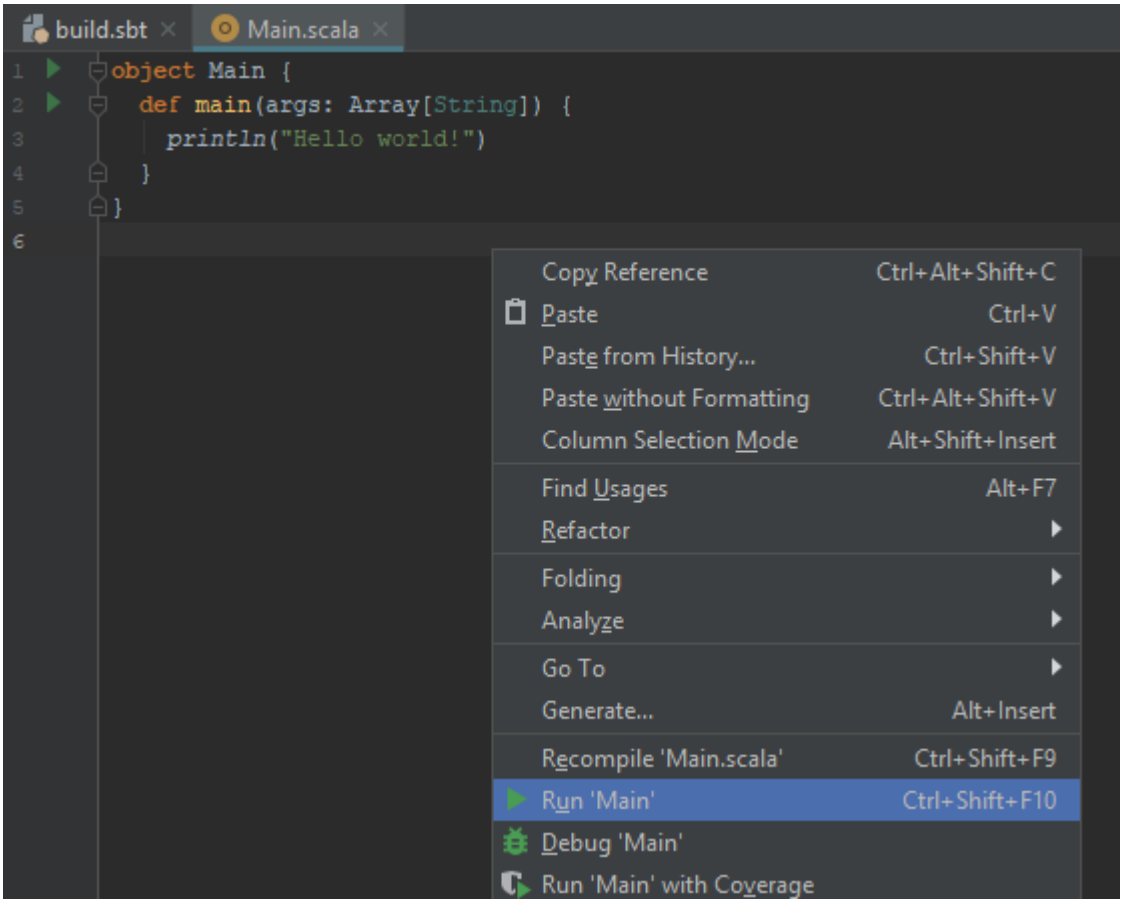




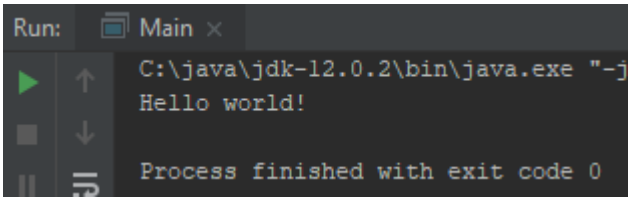
Примечание. Аналогом объекта object в Java является паттерн Singleton. Выполнения тела объекта происходит при его загрузке в память, аналогично инициализации в конструкторе, методы object доступны без создания объекта оператором new, аналогично публичным статическим методам.

```
object Main {
    def main(args: Array[String]) {
        println("Hello world")
    }
}
```

В контекстном меню выберите Run 'Main', либо нажмите сочетание клавиш Ctrl+Shift+F10.



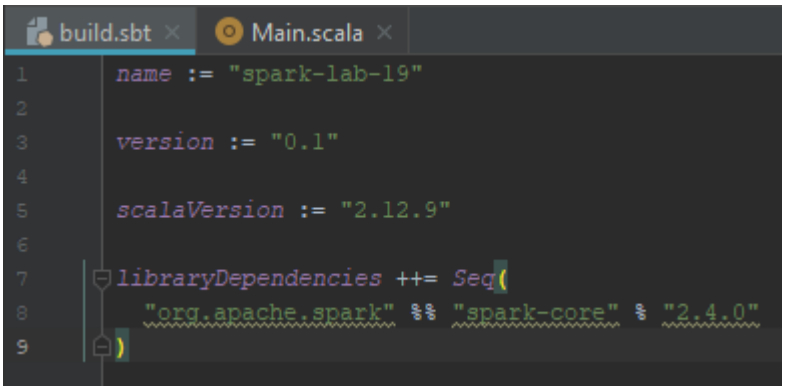
После выполнения в консоли должно появиться приветствие.



Добавьте к проекту зависимость Spark (версия на MapR кластере), записав следующие строки в конце файла build.sbt:

```
libraryDependencies ++= Seq(
    "org.apache.spark" %% "spark-core" % "2.4.0"
)
```

Сохраните изменения и обновите проект.



Подождите, когда SBT скачает все зависимые библиотеки.

Измените код Main.scala и создайте простейшую Spark программу. Импортируйте классы пакета *org.apache.spark*.

```
import org.apache.spark._
```

Создайте конфигурацию Spark с помощью класса SparkConf. Укажите обязательные параметры: имя запускаемой задачи (имя контекста задачи) и режим запуска (список режимов http://spark.apache.org/docs/latest/submitting-applications.html#master-urls). В нашем случае в качестве режима будет указан параметр local[2], означающий запуск с двумя потоками на локальной машине. В качестве режима может быть указан адрес главного узла.

```
val cfg = new SparkConf()
```

```
        .setAppName("Test").setMaster("local[2]")
```

Примечание. В Scala различаются два вида переменных: val и var. Переменные val являются неизменяемыми и инициализируются один раз, в отличие от var, которой можно присваивать новые значения несколько раз.

Инициализируйте контекст Spark в главном методе.

```
val sc = new SparkContext(cfg)
```

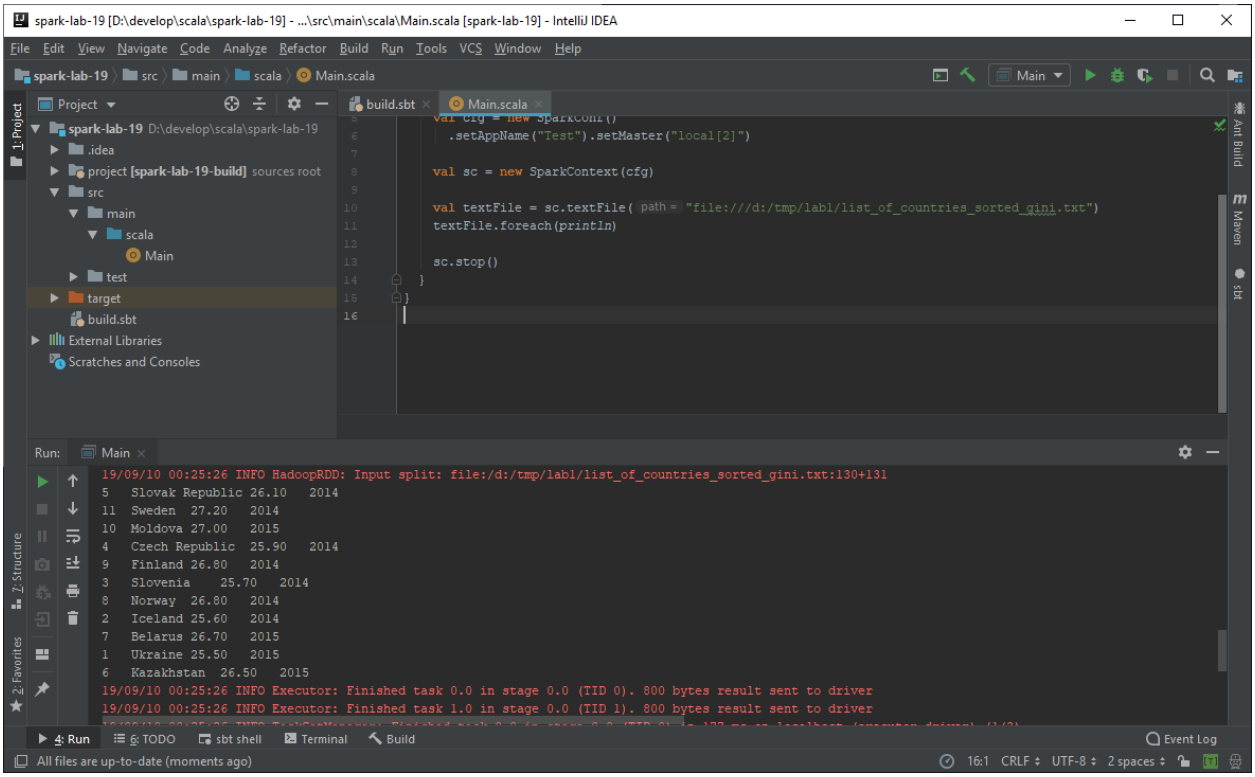
Добавьте в конец файла команду остановки контекста

```
sc.stop()
```

После инициализации контекста вы можете обращаться к командам Spark. Считайте любой текстовый файл из локальной файловой системы и выведите его по строкам в консоль.

Примечание. Путь к файлу в локальной файловой системе имеет определённый формат, имеющий префикс "file:///". https://tools.ietf.org/html/rfc8089

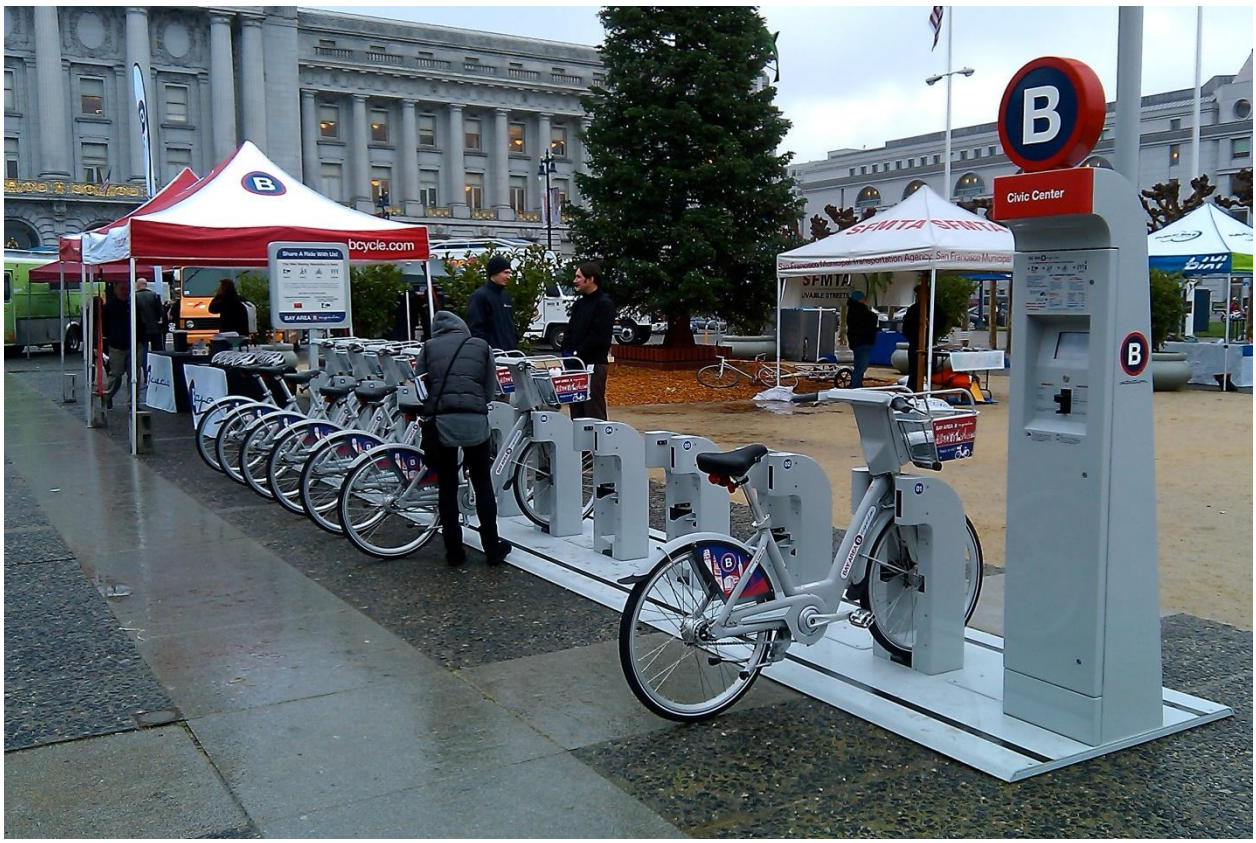
```
val textFile = sc.textFile("file:///c:/temp/file.txt")
textFile.foreach(println)
```



Примечание. При работе без winutils.exe запись в файловую систему будет порождать ошибку. Известным решением является скачивание данного файла из проекта Hadoop в файловую систему в папку с названием bin и указанием переменной Spark hadoop.home.dir. В переменной hadoop.home.dir хранится путь к папке с Hadoop определённой версии. Установить переменную среды JVM вы можете кодом System.setProperty(key, value) . Другим решением проблемы является установка переменной среды HADOOP_HOME (потребуется перезапуск IDE).
https://issues.apache.org/jira/browse/SPARK-2356.

Анализ данных велопарковок

Тестовыми данными являются список поездок на велосипедах trips.csv и список велостоянок проката велосипедов stations.csv.



Создайте по одному RDD на основе каждого файла stations.csv, trips.csv. Считайте данные в переменную, затем запомните заголовок. Объявите новую переменную с данными, в которых не будет заголовка, а строки преобразованы в массивы строк в соответствии с разделителем — запятая.

Примечание. Существует более эффективный, но громоздкий способ исключить заголовок из данных с использованием метода mapPartitionWithIndex. Пример присутствует в первой части лабораторной работы в разделе нахождения десяти популярных номеров такси.

```
val tripData = sc.textFile("file:///z:/data/trips.csv")

// запомним заголовок, чтобы затем его исключить
val tripsHeader = tripData.first
val trips = tripData.filter(row=>row!=tripsHeader)
            .map(row=>row.split(",",-1))

val stationData = sc.textFile("file:///z:/data/stations.csv")
```

```
val stationsHeader = stationData.first
val stations = stationData.filter(row=>row!=stationsHeader)
               .map(row=>row.split(",",-1))
```

Примечание. Использование в качестве второго параметра -1 в row.split(",",-1) позволяет не отбрасывать пустые строки. Например "1,2,".split(",") вернёт Array("1","2"), а "1,2,".split(",",-1) вернёт Array("1","2","").

Выведите заголовки таблиц и изучите колонки csv файлов.

```
stationsHeader
tripsHeader
```

Выведите несколько элементов данных в trips и stations.

Примечание. Убрать информационные строки логов из выдачи можно следующим образом:

```
import org.apache.log4j.{Logger, Level}
Logger.getLogger("org.apache.spark").setLevel(Level.WARN)
Logger.getLogger("org.spark-project").setLevel(Level.WARN)
```

Объявите stationsIndexed так, чтобы результатом был список пар ключ-значение с целочисленным ключом из первой колонки. Таким образом вы создаёте индекс на основе первой колонки - номера велостоянки

```
val stationsIndexed = stations.keyBy(row=>row(0).toInt)
```

Примечание. Обращение к массиву в Scala производится в круглых скобках. Например Array(1,2,3)(0) вернёт первый элемент.

Выведите часть данных нового RDD.

Аналогичное действие проделайте для индексирования коллекции trips по колонкам Start Terminal и End Terminal и сохраните результат в переменные, например tripsByStartTerminals и tripsByEndTerminals.

Выполните операцию объединения коллекций по ключу с помощью функции join. Объедините stationsIndexed и tripsByStartTerminals, stationsIndexed и tripsByEndTerminals.

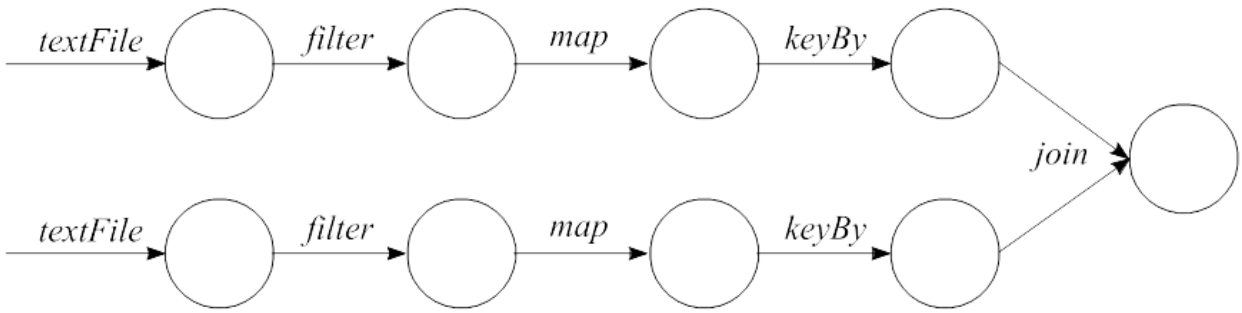
```
val startTrips =
stationsIndexed.join(tripsByStartTerminals)
```

```
val endTrips =
stationsIndexed.join(tripsByEndTerminals)
```

Объявление последовательности трансформаций приводит к созданию ацикличного ориентированного графа. Вывести полученный граф можно для любого RDD.

```
startTrips.toDebugString
endTrips.toDebugString
```

```
scala> startTrips.toDebugString
res39: String =
(2) MapPartitionsRDD[18] at join at <console>:20 []
 |  MapPartitionsRDD[17] at join at <console>:20 []
 |  CoGroupedRDD[16] at join at <console>:20 []
 +-(2) MapPartitionsRDD[10] at keyBy at <console>:15 []
 |  |  MapPartitionsRDD[9] at map at <console>:14 []
 |  |  MapPartitionsRDD[8] at filter at <console>:14 []
 |  |  MapPartitionsRDD[3] at textFile at <console>:12 []
 |  |  file://d:/station_data.csv HadoopRDD[2] at textFile at <console>:12 []
 +-(2) MapPartitionsRDD[14] at keyBy at <console>:15 []
    |  MapPartitionsRDD[7] at map at <console>:14 []
    |  MapPartitionsRDD[6] at filter at <console>:14 []
    |  MapPartitionsRDD[5] at textFile at <console>:12 []
    |  file://d:/trip_data.csv HadoopRDD[4] at textFile at <console>:12 []
```



Выполните объявленные графы трансформаций вызовом команды count.

```
startTrips.count()
endTrips.count()
```

Если вы знаете распределение ключей заранее, вы можете выбрать оптимальный способ хеширования ключей по разделам Partition. Например, если один ключ встречается на порядки чаще, чем другие ключи, то использование HashPartitioner будет не лучшим выбором, так как данные связанные с этим ключом будут собираться в одном разделе. Это приведёт к неравномерной нагрузке на вычислительные ресурсы.

Выбрать определённую реализацию класса распределения по разделам можно с помощью функции RDD partitionBy. Например, для RDD stationsIndexed выбирается HashPartitioner с количеством разделов равным количеству разделов trips RDD.

```
stationsIndexed.partitionBy(new
HashPartitioner(trips.partitions.size))
```

Также можно создавать свои классы для распределения ключей. Узнать какой класс назначен для текущего RDD можно обращением к полю partitioner.

```
stationsIndexed.partitioner
```

Создание модели данных

Для более эффективной обработки и получения дополнительных возможностей мы можем объявить классы сущностей предметной области и преобразовать исходные строковые данные в объявленное представление.

В Scala часто для объявления структур данных используется конструкция case class. Особенностью такого объявления класса являются: автоматическое создание методов доступа get для аргументов конструктора, автоматическое определение методов hashcode и equals, возможность case классов быть разобранными по шаблону (pattern matching). Например, для определения

```
case class IntNumber(val value:Integer)
```

выполнение

```
new IntNumber(4).value
```

вернёт значение 4.

Объявите case классы для представления строк таблиц в соответствии с именами заголовков.

```
case class Station(
  stationId:Integer,
  name:String,
  lat:Double,
  long:Double,
  dockcount:Integer,
  landmark:String,
  installation:String,
  notes:String)

case class Trip(
  tripId:Integer,
  duration:Integer,
  startDate:LocalDateTime,
  startStation:String,
  startTerminal:Integer,
  endDate:LocalDateTime,
  endStation:String,
  endTerminal:Integer,
  bikeId: Integer,
  subscriptionType: String,
```

```
  zipCode: String)
```

Для конвертации времени будем использовать пакет java.time. Краткое введение в работу с пакетом находится в Приложении Б. Объявим формат данных.

```
val timeFormat = DateTimeFormatter.ofPattern("M/d/yyyy
H:m")
```

Объявим trips с учётом преобразования во внутреннее представление.

```
val tripsInternal = trips.mapPartitions(rows => {
  val timeFormat =
DateTimeFormatter.ofPattern("M/d/yyyy H:m")
  rows.map( row =>
    new Trip(tripId=row(0).toInt,
      duration=row(1).toInt,
      startDate= LocalDate.parse(row(2), timeFormat),
      startStation=row(3),
      startTerminal=row(4).toInt,
      endDate=LocalDate.parse(row(5), timeFormat),
      endStation=row(6),
      endTerminal=row(7).toInt,
      bikeId=row(8).toInt,
      subscriptionType=row(9),
      zipCode=row(10)))})
```

Изучите полученные данные. Например, вызовом следующих команд

```
tripsInternal.first
tripsInternal.first.startDate
```

Примечание. В связи с тем, что timeFormat содержит несериализуемый объект, его необходимо создавать на каждом узле для каждой партиции.

То же можно проделать и для station RDD

```
val stationsInternal = stations.map(row=>
    new Station(stationId=row(0).toInt,
                name=row(1),
                lat=row(2).toDouble,
                long=row(3).toDouble,
                dockcount=row(4).toInt,
                landmark=row(5),
                installation=row(6)
```

```
                notes=null))
```

Примечание. Восьмая колонка не присутствует в таблице, так как в данных она пустая. Если в будущем она не будет использоваться, имеет смысл её убрать из описания case класса.

Примечание. В данных присутствуют различные форматы времени.

Посчитаем среднее время поездки, используя *groupByKey.*

Для этого потребуется преобразовать trips RDD в RDD коллекцию пар ключ-значение аналогично тому, как мы совершали это ранее методом *keyBy*.

```
val tripsByStartStation = tripsInternal.keyBy(record =>
record.startStation)
```

Рассчитаем среднее время поездки для каждого стартового парковочного места

```
val avgDurationByStartStation = tripsByStartStation
 .mapValues(x=>x.duration)
 .groupByKey()
 .mapValues(col=>col.reduce((a,b)=>a+b)/col.size)
```

Выведем первые 10 результатов

```
avgDurationByStartStation.take(10).foreach(println)
```

Выполнение операции groupByKey приводит к интенсивным передачам данных. Если группировка делается для последующей редукции элементов лучше использовать трансформацию reduceByKey или aggregateByKey. Их выполнение приведёт сначала к локальной редукции над разделом Partition, а затем будет произведено окончательное суммирование над полученными частичными суммами.

Примечание. Выполнение reduceByKey логически сходно с выполнением Combine и Reduce фазы MapReduce работы.

Функция aggregateByKey является аналогом reduceByKey с возможностью указывать начальный элемент.

Рассчитаем среднее значение с помощью aggregateByKey. Одновременно будут вычисляться два значения для каждого стартового терминала: сумма времён и количество поездок.

```
val avgDurationByStartStation2 = tripsByStartStation
  .mapValues(x=>x.duration)
  .aggregateByKey((0,0))(
   (acc, value) => (acc._1 + value, acc._2 + 1),
   (acc1, acc2) => (acc1._1+acc2._1, acc1._2+acc2._2))
  .mapValues(acc=>acc._1/acc._2)
```

В первых скобках передаётся начальное значение. В нашем случае это пара нулей. Первая анонимная функция предназначена для прохода по коллекции раздела. На этом проходе значение элементов помещаются средой в переменную value, а переменная «аккумулятора» acc накапливает значения. Вторая анонимная функция предназначена для этапа редукции частично посчитанных локальных результатов.

Сравните результаты avgDurationByStartStation и avgDurationByStartStation2 и их время выполнения.

Теперь найдём первую поездку для каждой велостоянки. Для решения опять потребуется группировка. Ещё одним недостатком groupByKey данных является то, что для группировки данные должны поместиться в оперативной памяти. Это может привести к ошибке OutOfMemoryException для больших объёмов данных.

Сгруппируем поездки по велостоянкам и отсортируем поездки в группах по возрастанию даты.

```
val firstGrouped = tripsByStartStation
  .groupByKey()
  .mapValues(x =>
    x.toList.sortWith((trip1, trip2) =>
      trip1.startDate.compareTo(trip2.startDate)<0))
```

```
(Mountain View City Hall,Trip(4081,218,2013-08-29T09:38,Mountain View City Hall,27,2013-08-29T09:41,Moun
(California Ave Caltrain Station,Trip(4375,880,2013-08-29T12:26,California Ave Caltrain Station,36,2013-
(San Jose Civic Center,Trip(4510,166,2013-08-29T13:31,San Jose Civic Center,3,2013-08-29T13:34,San Salva
(Yerba Buena Center of the Arts (3rd @ Howard),Trip(4355,2044,2013-08-29T12:18,Yerba Buena Center of the
(Commercial at Montgomery,Trip(4086,178,2013-08-29T09:42,Commercial at Montgomery,45,2013-08-29T09:45,Co

scala>
```

Лучшим вариантом с точки зрения эффективности будет использование трансформации reduceByKey

```
val firstGrouped = tripsByStartStation
  .reduceByKey((trip1,trip2) =>
```

```
    if (trip1.startDate.compareTo(trip2.startDate)<0)
    trip1 else trip2)
```

В данном случае «передаваться дальше» будет меньшее из значений ключа.

```
(Mountain View City Hall,Trip(4081,218,2013-08-29T09:38,Mountain View City Hall,27,2013-08-29T09:41,Mo
(California Ave Caltrain Station,Trip(4375,880,2013-08-29T12:26,California Ave Caltrain Station,36,201
(San Jose Civic Center,Trip(4510,166,2013-08-29T13:31,San Jose Civic Center,3,2013-08-29T13:34,San Sal
(Yerba Buena Center of the Arts (3rd @ Howard),Trip(4355,2044,2013-08-29T12:18,Yerba Buena Center of t
```

Задачи:

1. Найти велосипед с максимальным пробегом.
2. Найти наибольшее расстояние между станциями.
3. Найти путь велосипеда с максимальным пробегом через станции.
4. Найти количество велосипедов в системе.
5. Найти пользователей потративших на поездки более 3 часов.

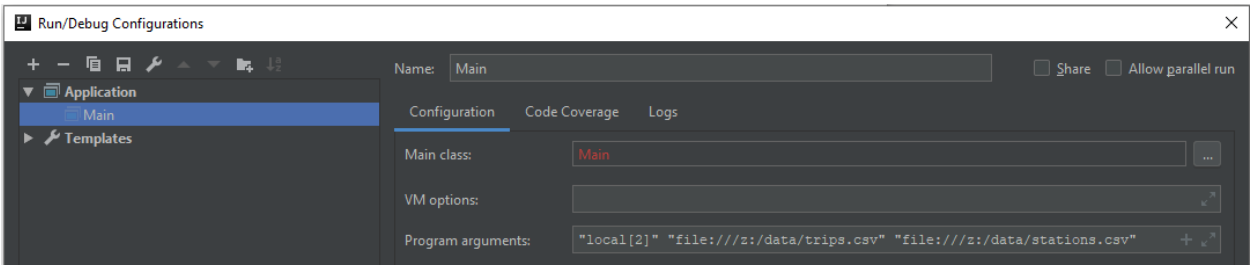
Запуск проекта на удалённом сервере

Для запуска собранного проекта на сервере используйте команду spark-submit. Однако прежде чем собрать проект, необходимо его изменить, так как в данный момент в коде зашиты пути к файлам с данными в локальной системе и режим запуска (setMaster("local[2]")).

Параметризуйте эти значения аргументами передаваемыми в программу при запуске.

```
val Seq(masterURL, tripDataPath, stationDataPath) = args.toSeq
val cfg = new
SparkConfig().setAppName("Test").setMaster(masterURL)

val tripData = sc.textFile(tripDataPath)
val stationData = sc.textFile(stationDataPath)
```

В конфигурации запуска добавьте значения аргументов:



Проверьте, что проект работает на локальном компьютере.

Соберите JAR с помощью sbt команды package. Файл появится в директории target/scala-2.12. Скопируйте его на сервер с помощью scp и запустите.

```
$ spark-submit --deploy-mode cluster untitled4_2.11-
0.1.jar yarn /labs/lab1/trips.csv
/labs/lab1/stations.csv
```

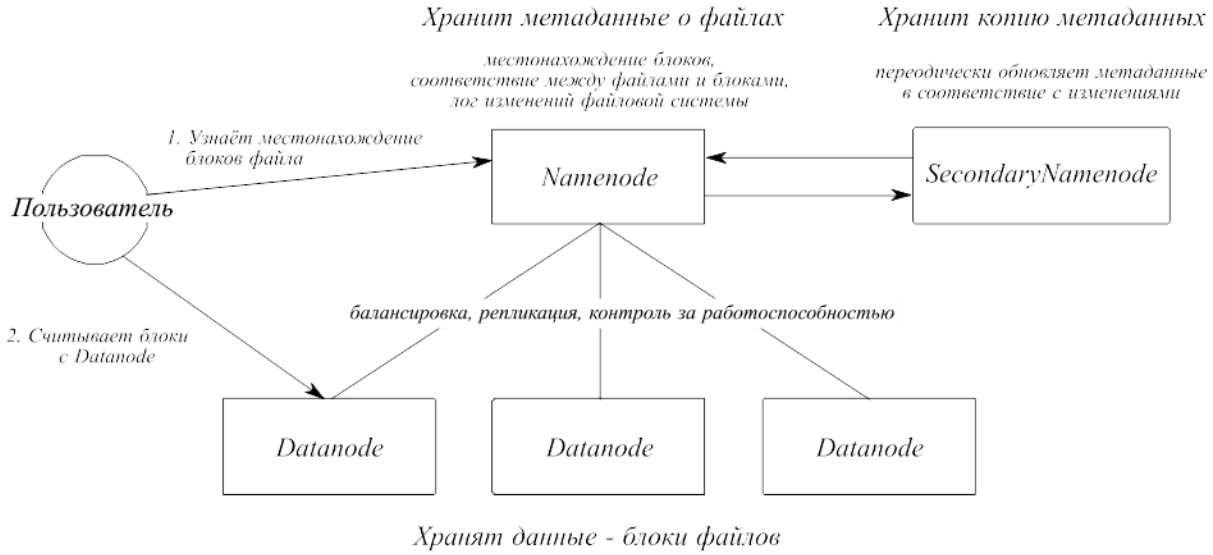
Логи YARN контейнеров вы можете найти в директории /mapr/tmp/studX/. Проверьте, что выдача вашей программы на сервере идентична выдаче в IDE при запуске на локальном компьютере.

Приложение А

Краткое описание файловой системы HDFS

HDFS — распределенная файловая система, используемая в проекте Hadoop. HDFS-кластер в первую очередь состоит из NameNode-сервера и DataNode-серверов, которые хранят данные. NameNode-сервер управляет пространством имен файловой системы и доступом клиентов к данным. Чтобы разгрузить NameNode-сервер, передача данных осуществляется только между клиентом и DataNode-сервером.

Архитектура HDFS



Развёртывание экземпляра HDFS предусматривает наличие центрального узла имён (англ. name node), хранящего метаданные файловой системы и метаинформацию о распределении блоков, и серии узлов данных (англ. data node), непосредственно хранящих блоки файлов. Узел имён отвечает за обработку операций уровня файлов и каталогов — открытие и закрытие файлов, манипуляция с каталогами, узлы данных непосредственно отрабатывают операции по записи и чтению данных. Узел имён и узлы данных снабжаются веб-серверами, отображающими текущий статус узлов и позволяющими просматривать содержимое файловой системы. Административные функции доступны из интерфейса командной строки.

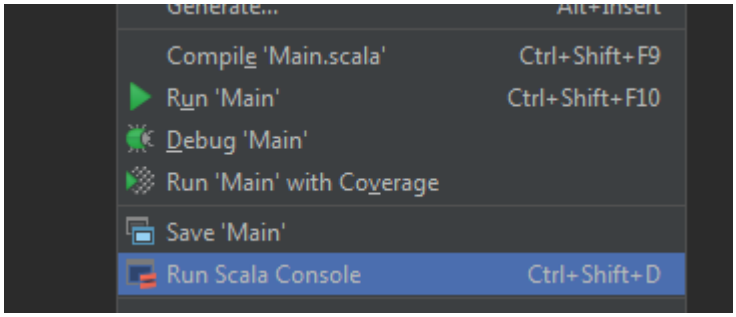
Приложение Б

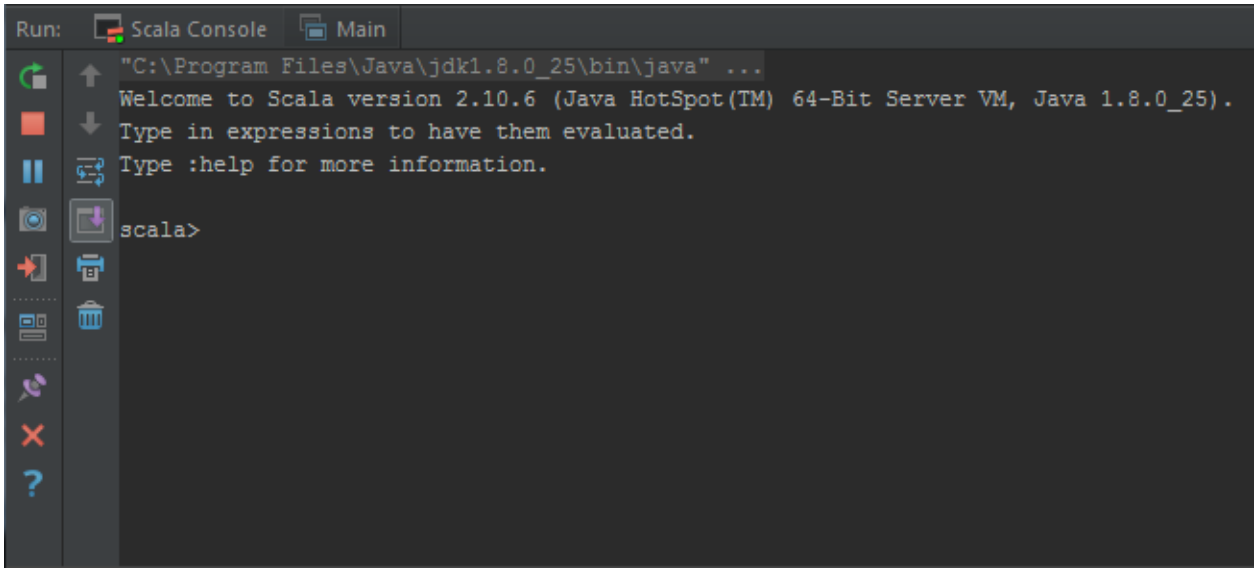
Основные понятия java.time

Для представления времени в Java 8 рекомендуется использовать пакет java.time, реализующий стандарт JSR 310. Документация пакета java.time доступна по адресу https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html.

Далее приводится работа с основными классами представления времени java.time. Для экспериментов удобно использовать REPL консоль. Если вы находитесь в среде разработки IDEA Scala консоль может быть запущена нажатием Ctrl+Shift+D, либо через контекстное меню Intellij IDEA.

Примечание. REPL **(**от сокращения read, eval, print, loop - считать, выполнить, напечатать, повторять в цикле) – интерактивный цикл взаимодействия программной среды с пользователем.





Примечание. Консоль также можно запустить в командном окне операционной системы с помощью sbt console, находясь в папке с проектом.

В обоих вариантах зависимости проекта подключаются автоматически так, что вы можете работать со сторонними библиотеками.

В пакете java.time различаются представления времени:

- класс Instant — хранит числовую метку;
- класс LocalDate — хранит дату без времени;
- класс LocalTime — хранит время без даты;
- класс LocalDateTime — хранит время и дату;
- класс ZonedDateTime — хранит дату, время и часовой пояс.

Узнайте в консоли текущее время, вызвав статический метод now() у каждого класса, изучите возвращаемое представление. Например,

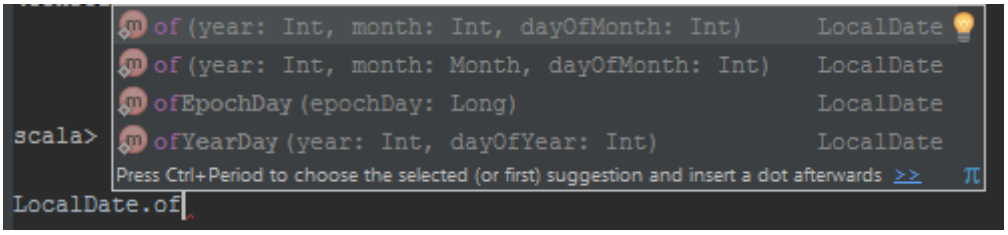
```
import java.time._
Instant.now()
```

Перед использованием классов объявите их импорт. Символ «_» импортирует все классы данного пакета.

Enter используется для переноса строки. Для выполнения нажмите сочетание клавиш Ctrl+Enter.

```
scala> import java.time._
Instant.now()
import java.time._

scala> res0: java.time.Instant = 2016-03-10T07:03:55.612Z
```

Создайте примечательную вам дату с помощью статического конструктора *of* классов LocalDate, LocalDateTime, ZonedDateTime. Воспользуйтесь подсказками среды разработки или документацией для определения количества аргументов метода *of* и их значения.




```
scala> LocalDate.of(2015,1,1)
res7: java.time.LocalDate = 2015-01-01
```

Изучите создание времён и дат с помощью метода *parse* данных классов. Используйте форматирование, которое выдавалось системой при возвращении значения в консоль.

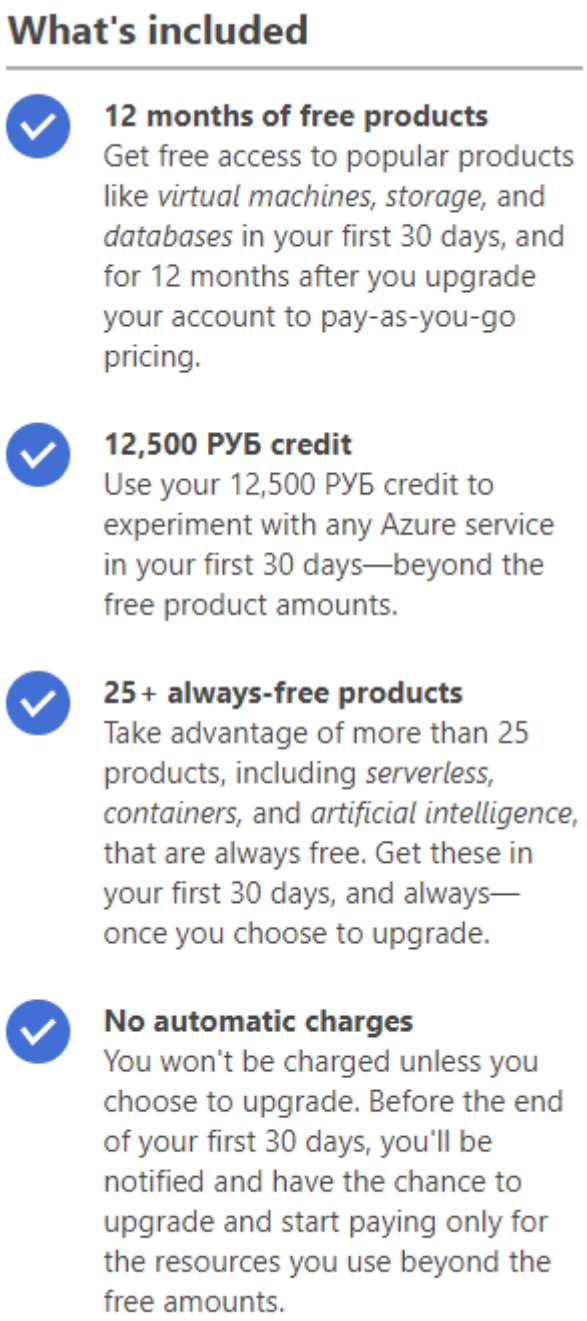
```
scala> LocalDate.parse("2015-09-01")
res8: java.time.LocalDate = 2015-09-01

scala> LocalTime.parse("00:00:00")
res9: java.time.LocalTime = 00:00

scala> LocalDateTime.parse("2015-09-01T00:30:00")
res10: java.time.LocalDateTime = 2015-09-01T00:30

scala> ZonedDateTime.parse("2015-09-01T00:00:00+04:00")
res11: java.time.ZonedDateTime = 2015-09-01T00:00+04:00
```

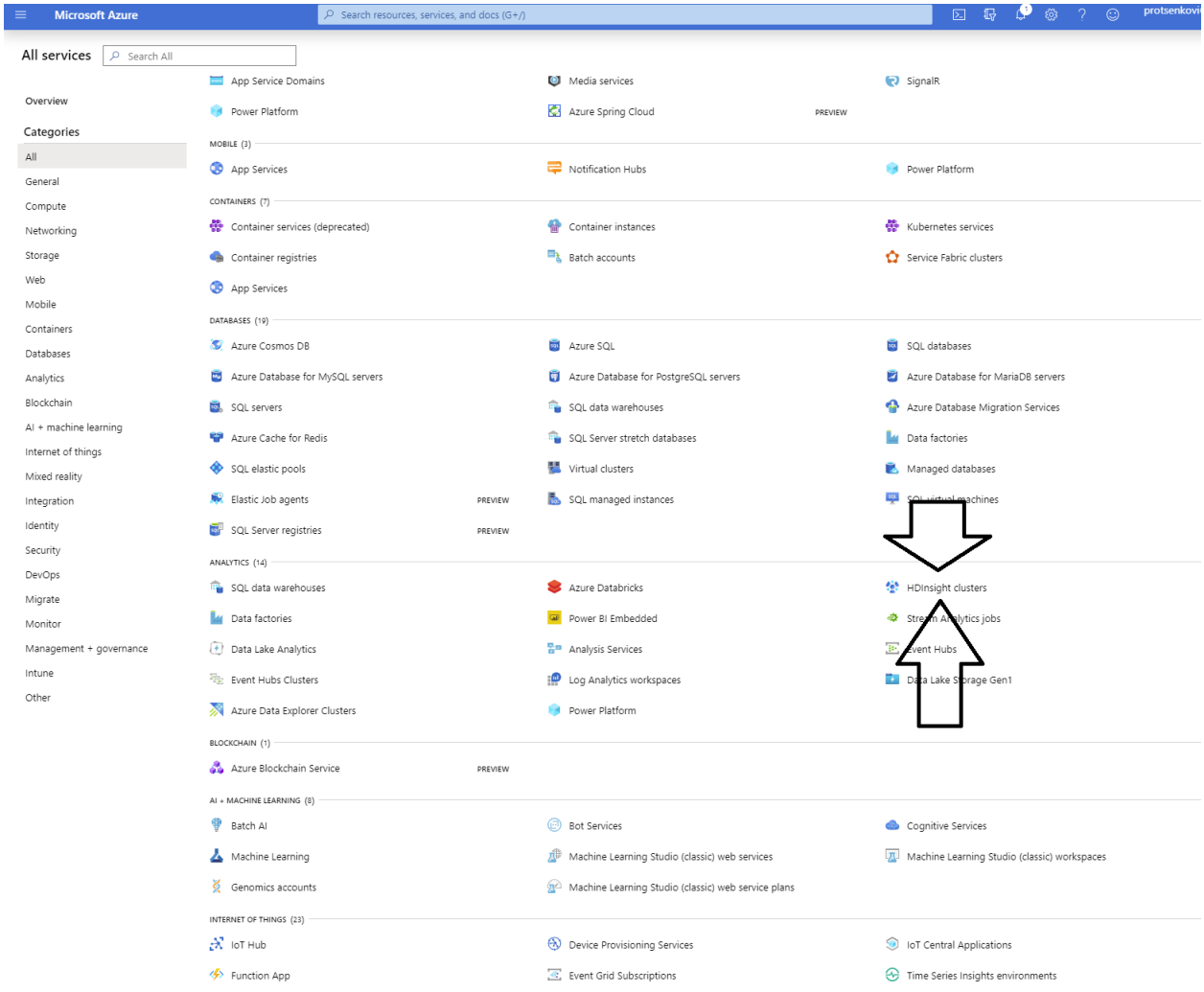
Для задания пользовательского формата считывания вы можете использовать класс DateTimeFormatter. Описание класса и символов шаблона располагается по адресу https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html.

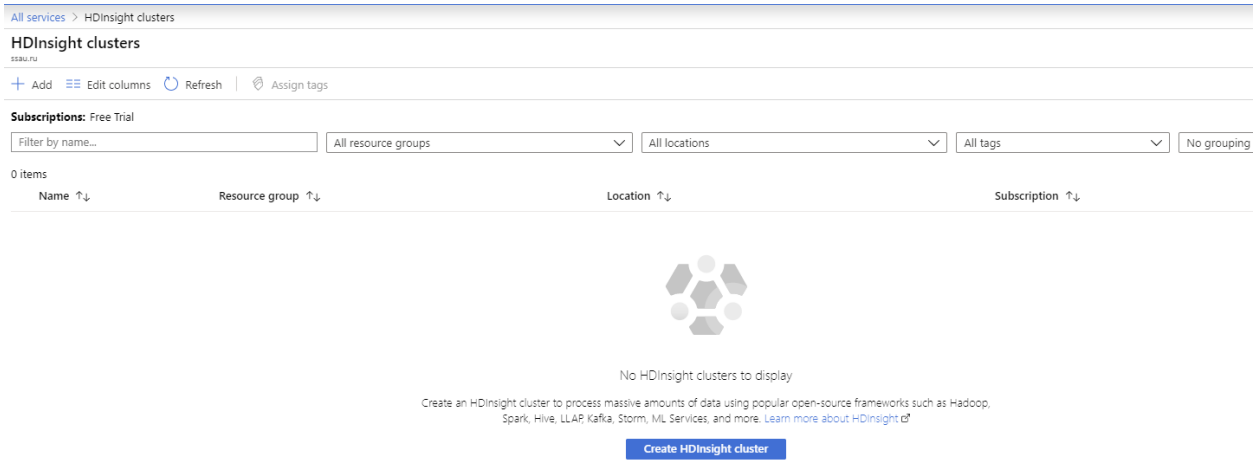
Приложение В

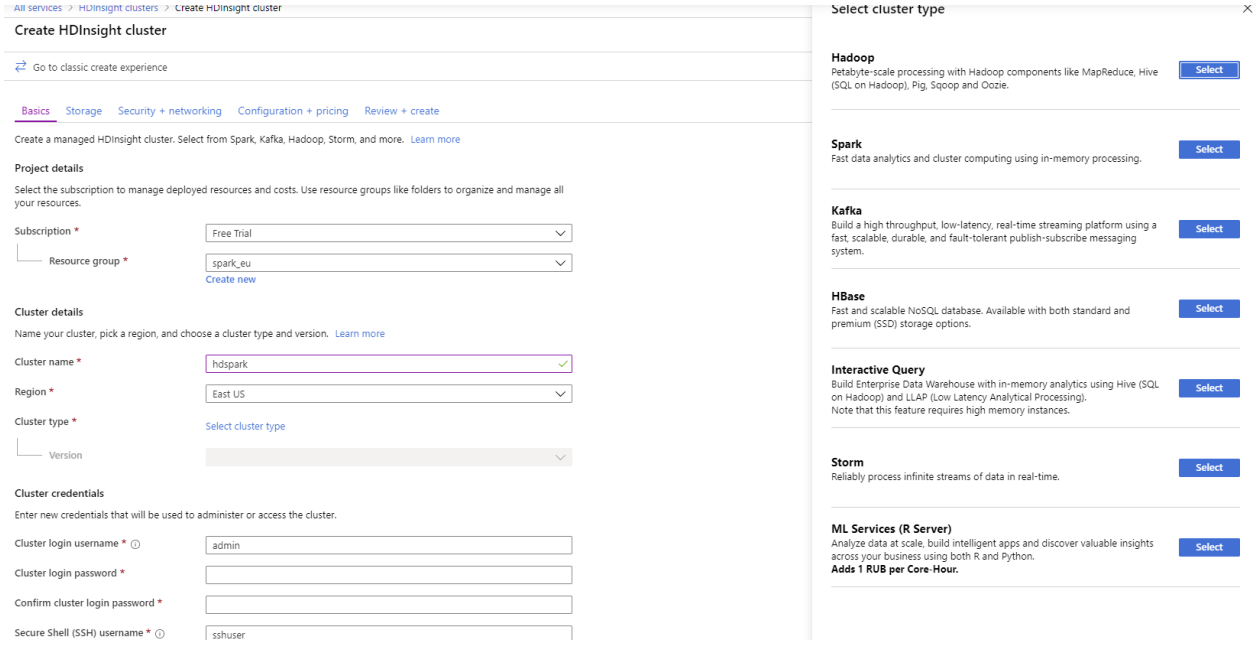
Настройка кластера в Microsoft Azure. При регистрации понадобится банковская карта. Для новых пользователей предоставляется 12 месяцев бесплатного использования и 12500 рублей на счету.

Ниже приведена последовательность шагов конфигурирования HDInsight кластера.




Microsoft Azure
Search resources, services, and docs (G+/)
All services
Search All
Overview
Categories
All
General
Compute
Networking
Storage
Web
Mobile
Containers
Databases
Analytics
Blockchain
AI + machine learning
Internet of things
Mixed reality
Integration
Identity
Security
DevOps
Migrate
Monitor
Management + governance
Intune
Other
App Service Domains
Media services
SignalR
Power Platform
Azure Spring Cloud
PREVIEW
MOBILE (3)
App Services
Notification Hubs
Power Platform
CONTAINERS (7)
Container services (deprecated)
Container instances
Kubernetes services
Container registries
Batch accounts
Service Fabric clusters
App Services
DATABASES (19)
Azure Cosmos DB
Azure SQL
SQL databases
Azure Database for MySQL servers
Azure Database for PostgreSQL servers
Azure Database for MariaDB servers
SQL servers
SQL data warehouses
Azure Database Migration Services
Azure Cache for Redis
SQL Server stretch databases
Data factories
SQL elastic pools
Virtual clusters
Managed databases
Elastic Job agents
PREVIEW
SQL managed instances
SQL virtual machines
SQL Server registries
PREVIEW
ANALYTICS (14)
SQL data warehouses
Azure Databricks
HDInsight clusters
Data factories
Power BI Embedded
Stream Analytics jobs
Data Lake Analytics
Analysis Services
Event Hubs
Event Hubs Clusters
Log Analytics workspaces
Data Lake Storage Gen1
Azure Data Explorer Clusters
Power Platform
BLOCKCHAIN (1)
Azure Blockchain Service
PREVIEW
AI + MACHINE LEARNING (8)
Batch AI
Bot Services
Cognitive Services
Machine Learning
Machine Learning Studio (classic) web services
Machine Learning Studio (classic) workspaces
Genomics accounts
Machine Learning Studio (classic) web service plans
INTERNET OF THINGS (23)
IoT Hub
Device Provisioning Services
IoT Central Applications
Function App
Event Grid Subscriptions
Time Series Insights environments


All services > HDInsight clusters
HDInsight clusters
Add
Edit columns
Refresh
Assign tags
Subscriptions: Free Trial
Filter by name...
All resource groups
All locations
All tags
No grouping
0 items
Name
Resource group
Location
Subscription
No HDInsight clusters to display
Create an HDInsight cluster to process massive amounts of data using popular open-source frameworks such as Hadoop, Spark, Hive, LLAP Kafka, Storm, ML Services, and more. Learn more about HDInsight
Create HDInsight cluster


All services > HDInsight clusters > Create HDInsight cluster
Create HDInsight cluster
Go to classic create experience
Basics
Storage
Security + networking
Configuration + pricing
Review + create
Create a managed HDInsight cluster. Select from Spark, Kafka, Hadoop, Storm, and more. Learn more
Project details
Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.
Subscription *
Free Trial
Resource group *
spark_eu
Create new
Cluster details
Name your cluster, pick a region, and choose a cluster type and version. Learn more
Cluster name *
hdspark
Region *
East US
Cluster type *
Select cluster type
Version
Cluster credentials
Enter new credentials that will be used to administer or access the cluster.
Cluster login username *
admin
Cluster login password *
Confirm cluster login password *
Secure Shell (SSH) username *
sshuser
Select cluster type
Hadoop
Petabyte-scale processing with Hadoop components like MapReduce, Hive (SQL on Hadoop), Pig, Sqoop and Oozie.
Select
Spark
Fast data analytics and cluster computing using in-memory processing.
Select
Kafka
Build a high throughput, low-latency, real-time streaming platform using a fast, scalable, durable, and fault-tolerant publish-subscribe messaging system.
Select
HBase
Fast and scalable NoSQL database. Available with both standard and premium (SSD) storage options.
Select
Interactive Query
Build Enterprise Data Warehouse with in-memory analytics using Hive (SQL on Hadoop) and LLAP (Low Latency Analytical Processing).
Note that this feature requires high memory instances.
Select
Storm
Reliably process infinite streams of data in real-time.
Select
ML Services (R Server)
Analyze data at scale, build intelligent apps and discover valuable insights across your business using both R and Python.
Adds 1 RUB per Core-Hour.
Select

Create HDInsight cluster

Go to classic create experience

Basics | Storage | Security + networking | Configuration + pricing | Review + create

Create a managed HDInsight cluster. Select from Spark, Kafka, Hadoop, Storm, and more. Learn more

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * : Free Trial

Resource group * : spark_eu

Create new

Cluster details

Name your cluster, pick a region, and choose a cluster type and version. Learn more

Cluster name * : hdspark

Region * : East US

Cluster type * : **Spark**

Change

Version * : Spark 2.4 (HDI 4.0)

Cluster credentials

Enter new credentials that will be used to administer or access the cluster.

Cluster login username * : admin

Cluster login password * : ••••••••••••••

Must contain at least one non-alphanumeric character, except characters ' " ` \).

Confirm cluster login password * : ••••••••••••••

Secure Shell (SSH) username * : sshuser

Use cluster login password for SSH

Create HDInsight cluster

Go to classic create experience

Basics | Storage | Security + networking | Configuration + pricing | Review + create

Select or create storage accounts that will be used for the cluster's logs, job input, and job output. Configure the cluster's access to these accounts, if needed.

Primary storage

Select or create a storage account that will be the default location for cluster logs and other output.

Primary storage type *: Azure Storage

Azure Storage

Selection method * ⓘ

Primary storage account *: (New) hdsparkhdistorage1

Create new

Container * ⓘ: hdspark-2019-12-25t17-44-32-227z

Data Lake Storage Gen1

Provide details for the cluster to access Data Lake Storage Gen1. The cluster will be able to access any Data Lake Storage Gen1 accounts that the chosen service principal has access to.

Data Lake Storage Gen1 access: Configure access settings

Additional Azure storage

Link additional Azure storage accounts to the cluster.

Add Azure storage

Metastore settings

To preserve your Hive and/or Oozie metadata outside of this cluster, select a SQL database for this cluster.

SQL database for Hive ⓘ

SQL database for Oozie ⓘ

SQL database for Ambari ⓘ

Create HDInsight cluster

Go to classic create experience

Basics | Storage | Security + networking | Configuration + pricing | Review + create

Configure cluster performance and pricing. Learn more

Node configuration

Configure your cluster's size and performance, and view estimated cost information.

The cost estimate represented in the table does not include subscription discounts or costs related to storage, networking, or data transfer.

This configuration will use 40 of 60 available cores in the East US region.
View cores usage

Add application

Node type	Node size	Number of ...	Estimated cost/hour
Head node	D12 v2 (4 Cores, 28 GB RAM), 23.36 RU...	2	46.73 RUB
Worker node	D13 v2 (8 Cores, 56 GB RAM), 46.73 RU...	4	186.90 RUB

Enable autoscale
Learn more

Total estimated cost/hour 233.63 RUB

Create HDInsight cluster

Go to classic create experience

Basics | Storage | Security + networking | Configuration + pricing | Review + create

Configure cluster performance and pricing. Learn more

Node configuration

Configure your cluster's size and performance, and view estimated cost information.

The cost estimate represented in the table does not include subscription discounts or costs related to storage, networking, or data transfer.

This configuration will use 24 of 60 available cores in the East US region.
View cores usage

Add application

Node type	Node size	Number of ...	Estimated cost/hour
Head node	D12 v2 (4 Cores, 28 GB RAM), 23.36 RU...	2	46.73 RUB
Worker node	D13 v2 (8 Cores, 56 GB RAM), 46.73 RU...	2	93.45 RUB

Enable autoscale
Learn more

Total estimated cost/hour 140.18 RUB

Basics | Storage | Security + networking | Configuration + pricing | Review + create

Spark 2.4 (HDI 4.0)

140.18 RUB Total estimated cost/hour

This estimate does not include subscription discounts or costs related to storage, networking, or data transfer.

Basics

Subscription	Free Trial
Resource group	spark_eu
Region	East US
Cluster name	(new) hdspark
Cluster type	Spark 2.4 (HDI 4.0)
Cluster login username	admin
Secure Shell (SSH) username	sshuser
Use cluster login password for SSH	Enabled

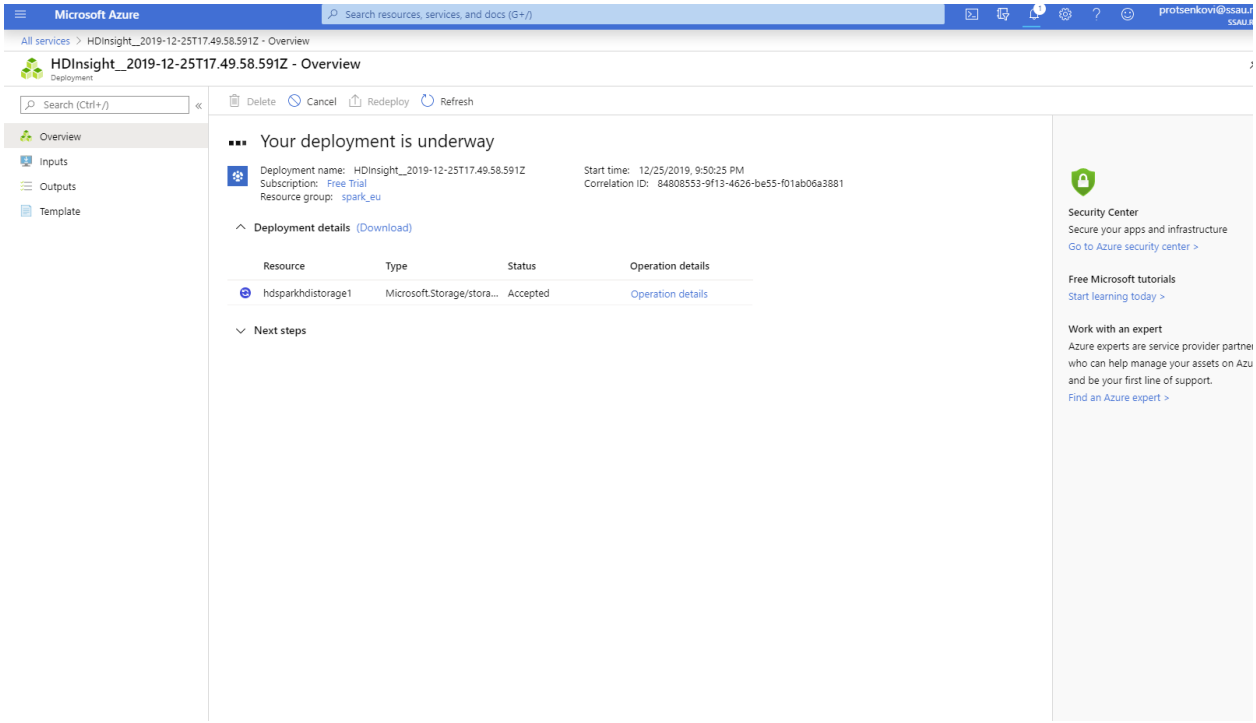
Storage

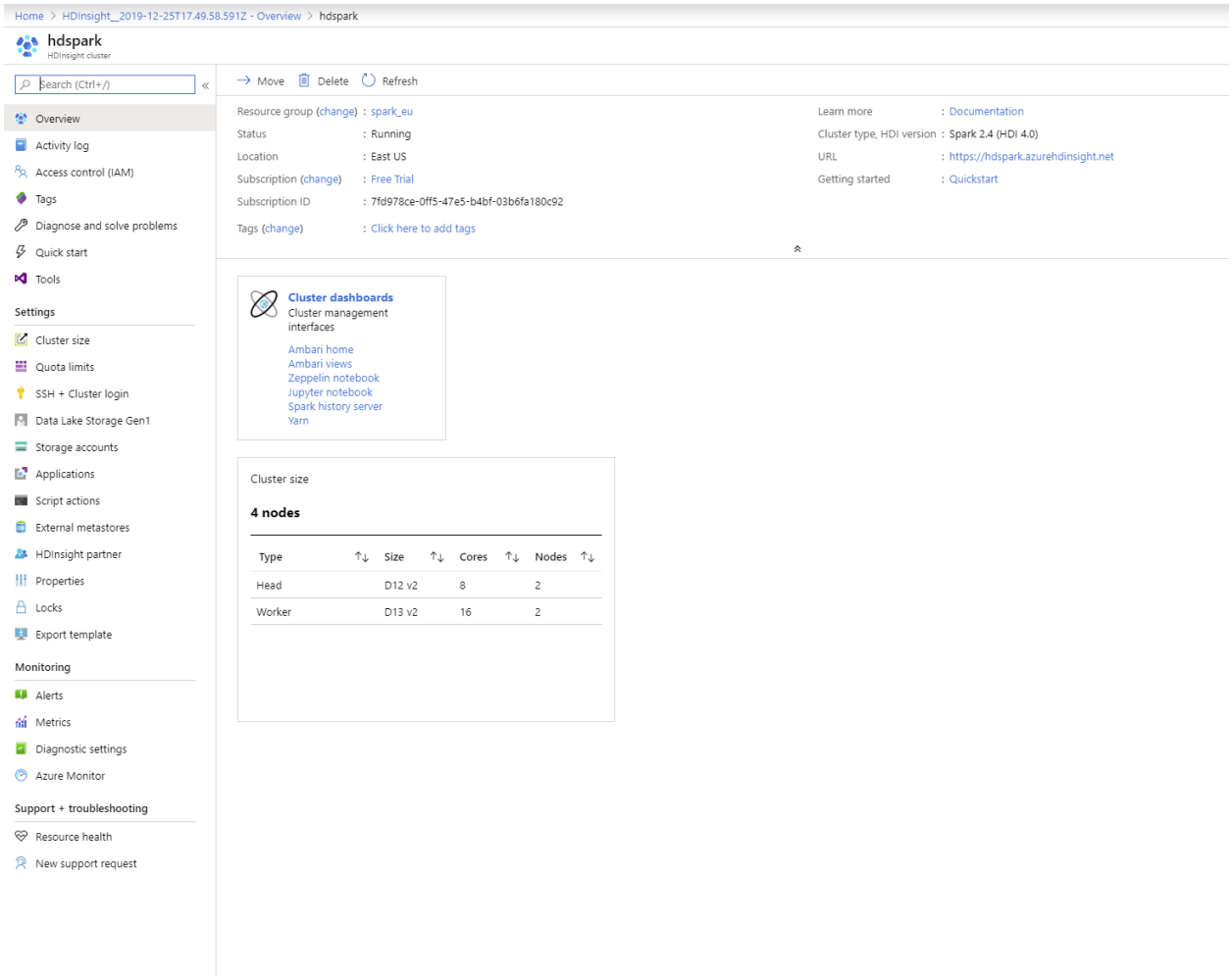
Primary storage type	Azure Storage
Primary storage account	(new) hdsparkhdistorage1
Container	hdspark-2019-12-25t17-44-32-227z
Additional Azure storage	None
Data Lake Storage Gen1 access	Disabled

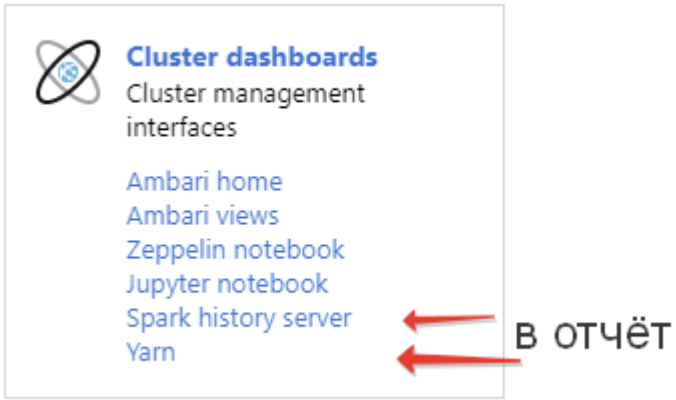
Cluster configuration

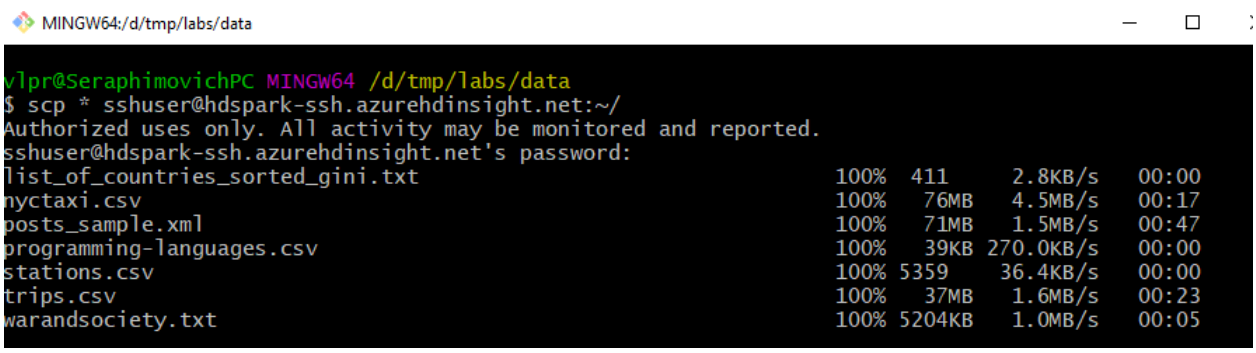
Head	2 nodes, D12 v2 (4 Cores, 28 GB RAM)
Worker	2 nodes, D13 v2 (8 Cores, 56 GB RAM)

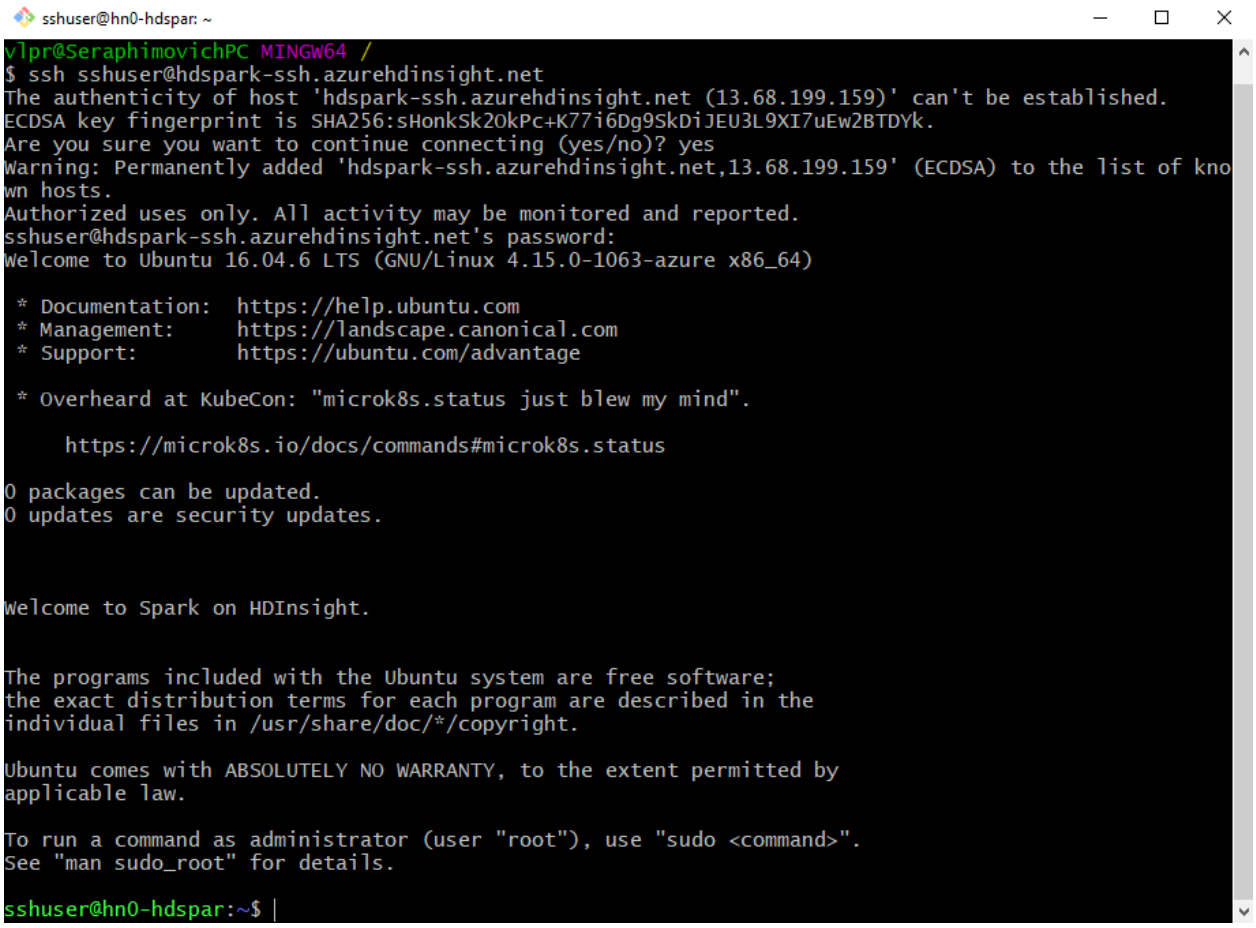
Create | « Previous | Next | Download a template for automation

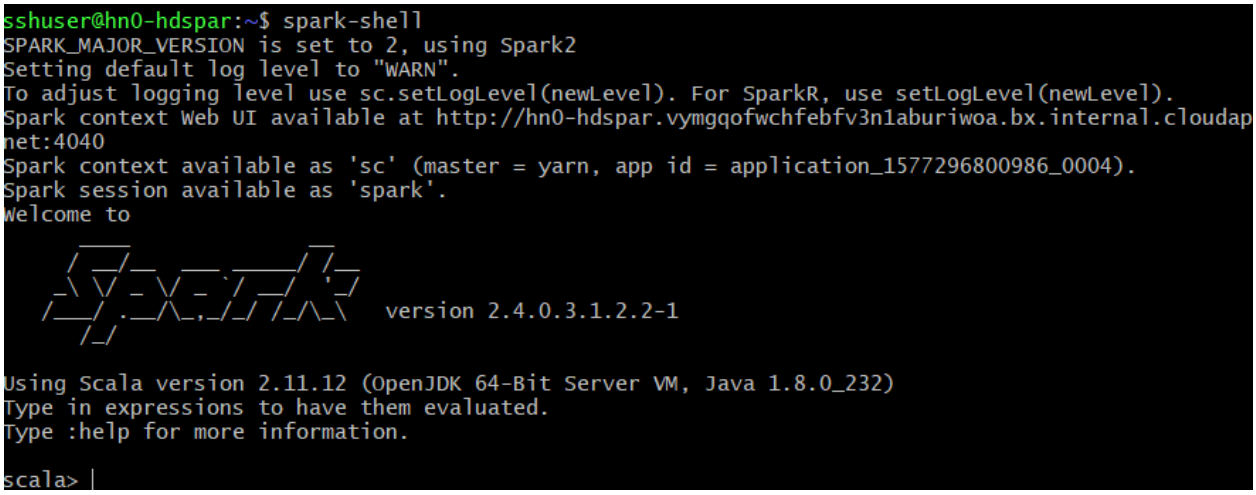

Microsoft Azure
Search resources, services, and docs (G+/)
protsenkovi@ssau.r
SSAU.R
All services > HDInsight__2019-12-25T17.49.58.591Z - Overview
HDInsight__2019-12-25T17.49.58.591Z - Overview
Deployment
Search (Ctrl+/)
Overview
Inputs
Outputs
Template
Delete
Cancel
Redeploy
Refresh
Your deployment is underway
Deployment name: HDInsight__2019-12-25T17.49.58.591Z
Subscription: Free Trial
Resource group: spark_eu
Start time: 12/25/2019, 9:50:25 PM
Correlation ID: 84808553-9f13-4626-be55-f01ab06a3881
Deployment details (Download)
Resource
Type
Status
Operation details
hdsparkhdistorage1
Microsoft.Storage/stora...
Accepted
Operation details
Next steps
Security Center
Secure your apps and infrastructure
Go to Azure security center >
Free Microsoft tutorials
Start learning today >
Work with an expert
Azure experts are service provider partner
who can help manage your assets on Azu
and be your first line of support.
Find an Azure expert >


Home > HDInsight_2019-12-25T17.49.58.591Z - Overview > hdspark
hdspark
HDInsight cluster
Search (Ctrl+/)
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Quick start
Tools
Settings
Cluster size
Quota limits
SSH + Cluster login
Data Lake Storage Gen1
Storage accounts
Applications
Script actions
External metastores
HDInsight partner
Properties
Locks
Export template
Monitoring
Alerts
Metrics
Diagnostic settings
Azure Monitor
Support + troubleshooting
Resource health
New support request
Move
Delete
Refresh
Resource group (change) : spark_eu
Status : Running
Location : East US
Subscription (change) : Free Trial
Subscription ID : 7fd978ce-0ff5-47e5-b4bf-03b6fa180c92
Tags (change) : Click here to add tags
Learn more : Documentation
Cluster type, HDI version : Spark 2.4 (HDI 4.0)
URL : https://hdspark.azurehdinsight.net
Getting started : Quickstart
Cluster dashboards
Cluster management interfaces
Ambari home
Ambari views
Zeppelin notebook
Jupyter notebook
Spark history server
Yarn
Cluster size
4 nodes
Type
Size
Cores
Nodes
Head
D12 v2
8
2
Worker
D13 v2
16
2


Cluster dashboards
Cluster management interfaces
Ambari home
Ambari views
Zeppelin notebook
Jupyter notebook
Spark history server
Yarn
В ОТЧЁТ


MINGW64:/d/tmp/labs/data
vlpr@SeraphimovichPC MINGW64 /d/tmp/labs/data
$ scp * sshuser@hdspark-ssh.azurehdinsight.net:~/
Authorized uses only. All activity may be monitored and reported.
sshuser@hdspark-ssh.azurehdinsight.net's password:
list_of_countries_sorted_gini.txt 100% 411 2.8KB/s 00:00
nyctaxi.csv 100% 76MB 4.5MB/s 00:17
posts_sample.xml 100% 71MB 1.5MB/s 00:47
programming-languages.csv 100% 39KB 270.0KB/s 00:00
stations.csv 100% 5359 36.4KB/s 00:00
trips.csv 100% 37MB 1.6MB/s 00:23
warandsociety.txt 100% 5204KB 1.0MB/s 00:05


sshuser@hn0-hdspar: ~
vlpr@SeraphimovichPC MINGW64 /
$ ssh sshuser@hdspark-ssh.azurehdinsight.net
The authenticity of host 'hdspark-ssh.azurehdinsight.net (13.68.199.159)' can't be established.
ECDSA key fingerprint is SHA256:sHonkSk2OkPc+K77i6Dg9SkDiJEU3L9XI7uEw2BTDYk.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'hdspark-ssh.azurehdinsight.net,13.68.199.159' (ECDSA) to the list of kno
wn hosts.
Authorized uses only. All activity may be monitored and reported.
sshuser@hdspark-ssh.azurehdinsight.net's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-1063-azure x86_64)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
* Overheard at KubeCon: "microk8s.status just blew my mind".
https://microk8s.io/docs/commands#microk8s.status
0 packages can be updated.
0 updates are security updates.
Welcome to Spark on HDInsight.
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
sshuser@hn0-hdspar:~$


sshuser@hn0-hdspar:~$ spark-shell
SPARK_MAJOR_VERSION is set to 2, using Spark2
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://hn0-hdspar.vymgqofwchfebfv3n1aburiwoa.bx.internal.cloudap
net:4040
Spark context available as 'sc' (master = yarn, app id = application_1577296800986_0004).
Spark session available as 'spark'.
Welcome to
version 2.4.0.3.1.2.2-1
Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_232)
Type in expressions to have them evaluated.
Type :help for more information.
scala>