Введение в Scala

Андрей Кузнецов

27.10.2022

Структура курса

1. Введение в Большие Данные
2. Hadoop экосистема и MapReduce
3. SQL поверх больших данных
4. Инструменты визуализации при работе с Большими Данными
5. Введение в Scala
6. Модель вычислений Spark: RDD
7. Approximate алгоритмы для больших данных
8. Потоковая обработка данных (Kafka, Spark Streaming, Flink)
9. Гостевая лекция VK
10. Гостевая лекция VK

План занятия

1. Основные понятия мира JVM
2. Введение Scala
3. Breeze
4. Workshop

Основные понятия мира JVM

Java world

Java - строго типизированный объектно-ориентированный язык программирования общего назначения. Приложения на Java компилируются в специальный байт-код, которые исполняется с помощью виртуальной Java-машины (JVM).

Другие JVM языки:

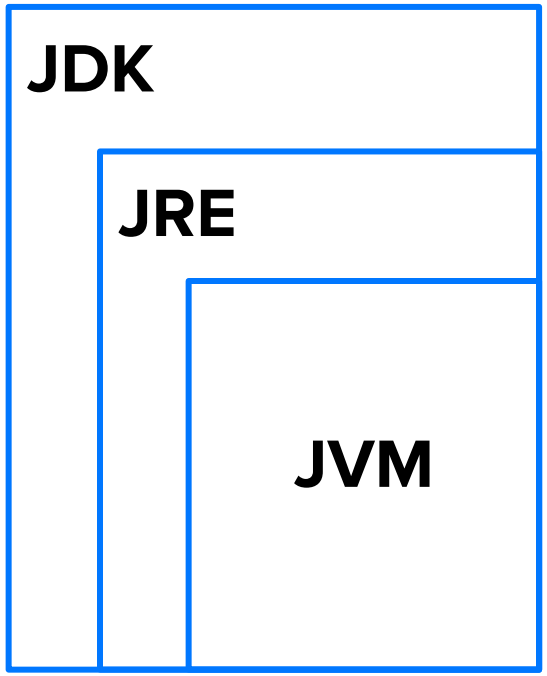
Clojure — функциональный язык, диалект Lisp (DT);
Groovy — сценарный язык (DT);
Kotlin — объектно-ориентированный язык для индустриальной разработки (ST);
Scala — объектно-ориентированный и функциональный язык (ST).

JVM JDK JRE javac

JVM (Java Virtual Machine) - виртуальная машина Java. Исполняет байт-код Java, предварительно созданный из исходного текста Java-программы компилятором Java (javac). JVM обеспечивает платформо-независимый способ выполнения кода.

JRE (Java Runtime Environment) - минимальная реализация виртуальной машины, необходимая для исполнения Java - приложений, без компилятора и других средств разработки. Состоит из виртуальной машины и библиотек Java классов.

JDK (Java Development Kit) - комплект разработчика приложений на языке Java, включающий в себя компилятор, стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную систему JRE.



Why it matters?

Большинство библиотек из стека бигдата написано на JVM языках:

- Hadoop
- Spark
- Kafka
- Ignite
- Samza/Flink
- Presto/Trino
- Cassandra
- Другие

Why we need Scala?

- Хороший старт в JVM-мире для DS, знающего Python благодаря простоте синтаксиса
- Лучшая поддержка Spark, так как он написан на Scala
- На порядки быстрее, чем Python при написании кастомной логики UDF

*https://habr.com/ru/company/odnoklassniki/blog/443324/

Введение в Scala

Scala in a nutshell

Scala — мультипарадигмальный язык программирования, спроектированный кратким и типобезопасным для простого и быстрого создания компонентного программного обеспечения, сочетающий возможности функционального и объектно-ориентированного программирования.

Scala похожа на Java и может свободно взаимодействовать с Java-кодом. Много синтаксического сахара по сравнению с Java. При этом:

- любое значение является объектом
- любая операция — вызовом метода

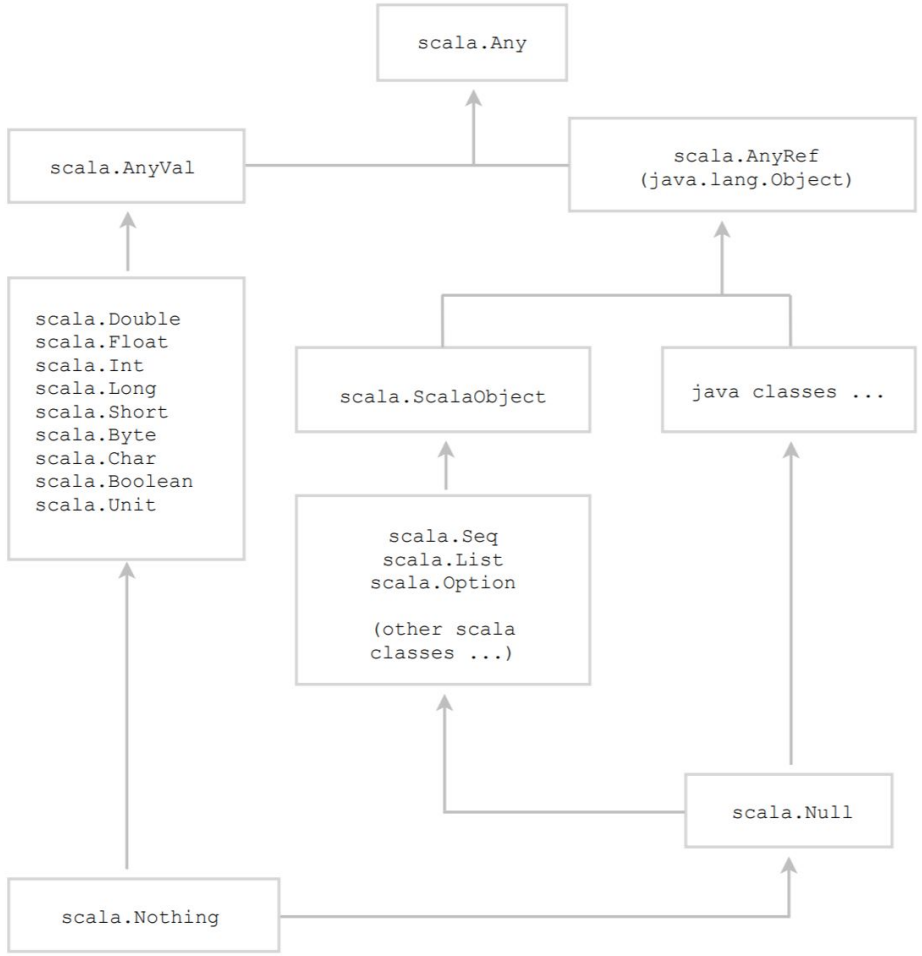
Примитивные типы Scala

Type	Values
`Byte`	`-128` to `127`
`Short`	`-32,768` to `32,767`
`Int`	`-2,147,483,648` to `2,147,483,647`
`Long`	`-9,223,372,036,854,775,808` to `9,223,372,036,854,775,807`

Type	Values
`Boolean`	`true`, `false`
`Char`	`'a'`, `'0'`, `'Z'`, `'包'`, ...
`Float`	32-bit Floating point
`Double`	64-bit Floating point

Иерархия типов Scala

- val значения и var переменные
- есть изменяемые (mutable) и неизменяемые (immutable) коллекции
- все типы наследуются обобщаются Any и делятся на типы значений AnyVal и ссылочные AnyRef
- кортежи - это набор значений фиксированной длины возможно разных типов



Функции и методы

Метод - определяется и работает как в Python

```
def helloMethod(input: String) = s“Hello, $input”
```

Функция - значение, которое может быть использовано как метод

```
val plusOne: Int = (x: Int) => x + 1
```

Управляющие конструкции

Классические управляющие конструкции, условия в скобках

```
if (foo) bar else baz
for (i <- 0 to 10) { ... }
while (true) { println("Hello, World!") }
```

for-comprehensions в наличии

```
for {
  x <- board.rows
  y <- board.files
} yield (x, y)
```

Тернарные операторы тоже

```
val res = if (foo) bar else baz
```

Option/Some/None

```
def toInt(s: String): Option[Int] = {
    try {
        Some(Integer.parseInt(s.trim))
    } catch {
        case e: Exception => None
    }
}
```

```
scala> val a = toInt("1")
a: Option[Int] = Some(1)

scala> val a = toInt("foo")
a: Option[Int] = None
```

Массивы

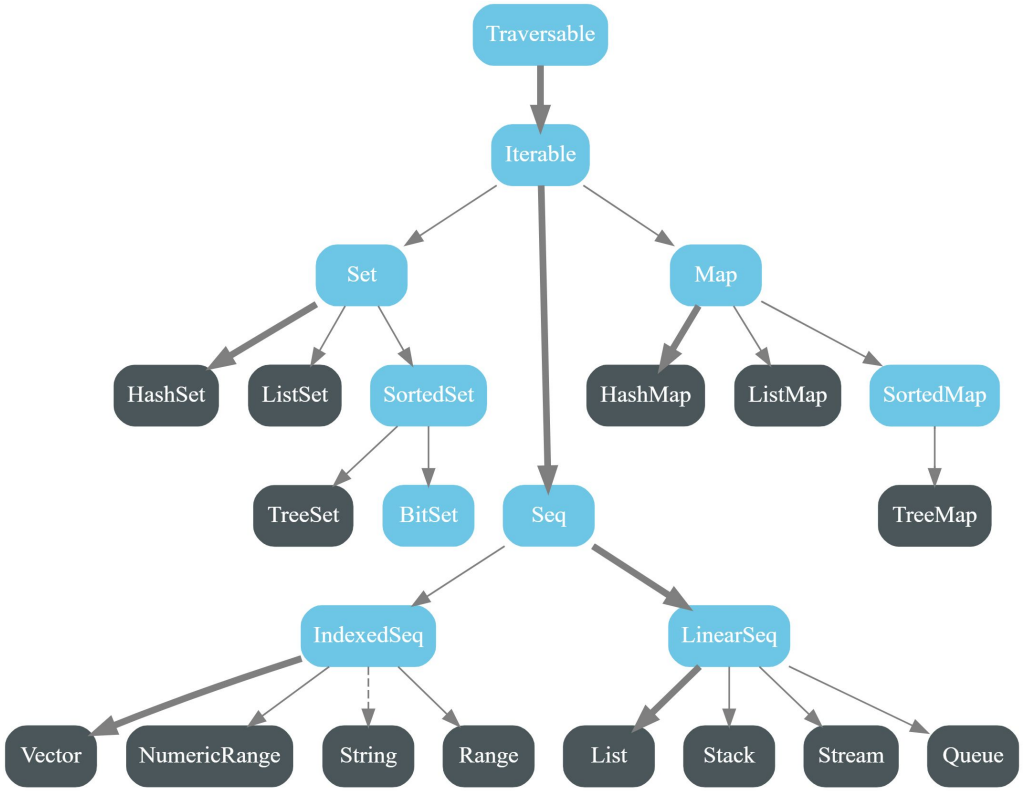
Scala массивы соответствуют массивам из Java. Например, Scala массив Array[Int] реализован в виде Java int[], а Array[Double] как Java double[] и Array[String] как Java String[].

Scala массивы совместимы со списками (Seq) Scala - вы можете передавать Array[T] на вход туда, где требуется Seq[T]. Ну и наконец, Scala массивы также поддерживают все операции, которые есть у списков.

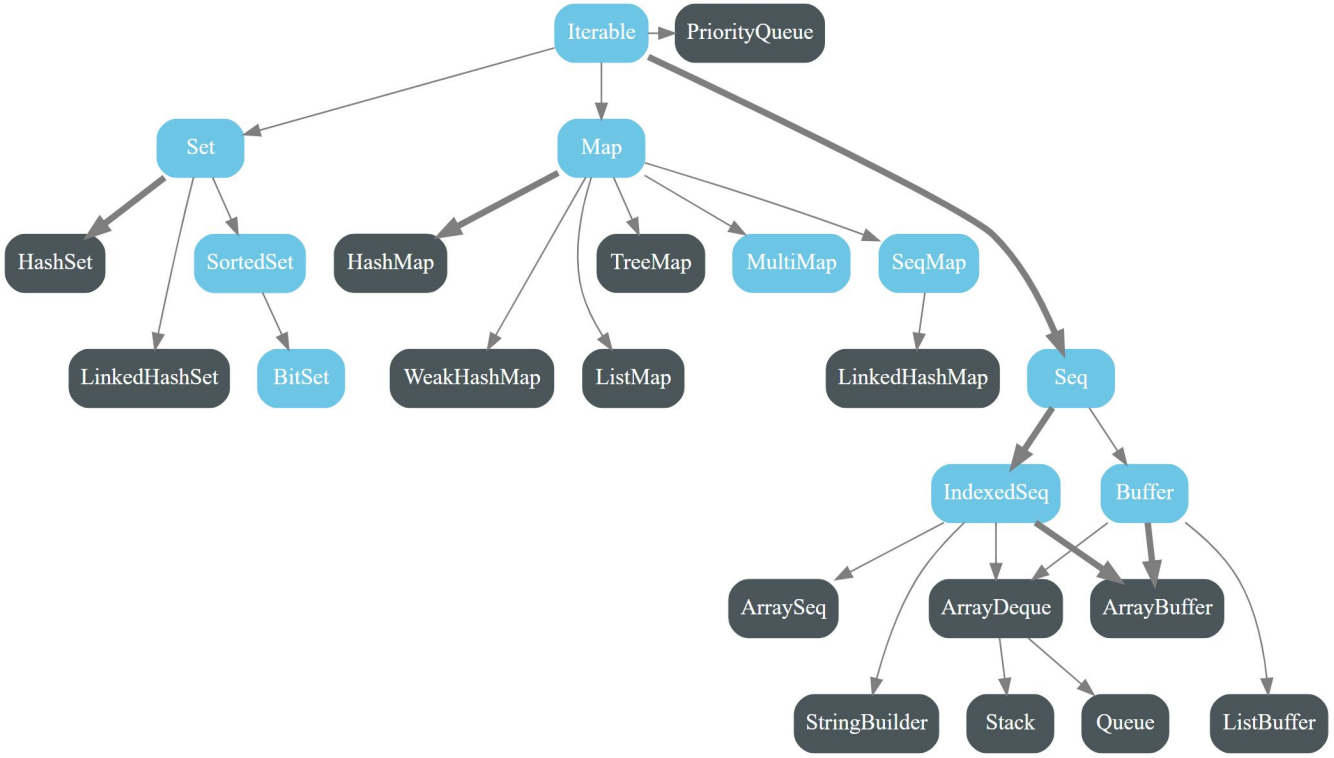
```
scala> val a1 = Array(1, 2, 3)
a1: Array[Int] = Array(1, 2, 3)
scala> val a2 = a1 map (_ * 3)
a2: Array[Int] = Array(3, 6, 9)
scala> val a3 = a2 filter (_ % 2 != 0)
a3: Array[Int] = Array(3, 9)
scala> a3.reverse
res0: Array[Int] = Array(9, 3)
```

Коллекции

Immutable



Mutable



Коллекции

	lookup	add	remove	min
Не изменяемые				
`HashSet` / `HashMap`	eC	eC	eC	L
`TreeSet` / `TreeMap`	Log	Log	Log	Log
`BitSet`	C	L	L	eC[1]
`VectorMap`	eC	eC	aC	L
`ListMap`	L	L	L	L
Изменяемые				
`HashSet` / `HashMap`	eC	eC	eC	L
`WeakHashMap`	eC	eC	eC	L
`BitSet`	C	aC	C	eC[1]
`TreeSet`	Log	Log	Log	Log

	head	tail	apply	update	prepend	append	insert
Не изменяемые							
`List`	C	C	L	L	C	L	-
`LazyList`	C	C	L	L	C	L	-
`ArraySeq`	C	L	C	L	L	L	-
`Vector`	eC	eC	eC	eC	eC	eC	-
`Queue`	aC	aC	L	L	C	C	-
`Range`	C	C	C	-	-	-	-
`String`	C	L	C	L	L	L	-
Изменяемые							
`ArrayBuffer`	C	L	C	C	L	aC	L
`ListBuffer`	C	L	L	L	C	C	L
`StringBuilder`	C	L	C	C	L	aC	L
`Queue`	C	L	L	L	C	C	L
`ArraySeq`	C	L	C	C	-	-	-
`Stack`	C	L	L	L	C	L	L
`Array`	C	L	C	C	-	-	-
`ArrayDeque`	C	L	C	C	aC	aC	L

https://docs.scala-lang.org/overviews/collections/performance-characteristics.html

Breeze

Breeze

Breeze - библиотека для вычислений на Scala. По методам и решаемым задачам похожа на Numpy, хотя и проигрывает в функциональности и поддержке сообщества.

Обычно нужно использовать, когда производительности и функциональности встроенных в Scala конструкции не хватает. В частности в векторных операциях.

Breeze. Data Structures

- `Vector`
  - DenseVector: a "normal" array-backed Vector.
  - SparseVector: a sparse vector backed by binary search. *O(log numNonZero)* access to elements with optimizations for in-order traversal. Increasing the number of nonzero elements in the SparseVector (say, via `update`) is expensive, as *O(numNonZero)*.
  - HashVector: a sparse vector backed by a quadratic-probing open address hash array. *O(1)* access, but typically less memory efficient than `SparseVector`. No in-order traversal.
- `Matrix`
  - DenseMatrix: a "normal" array-backed `Matrix`. Column-major unless isTranspose is true, in which case it is row-major. (This is consistent with standard BLAS/LAPACK assumptions.)
  - CSCMatrix: a still-under-development Compressed Sparse Columns Matrix. Each column of the matrix is analogous to a SparseVector: access to an element is *O(log numNonZeroInColumn)*.

Breeze methods

Creation

Operation	Breeze	Matlab	Numpy
Zeroed matrix	`val c = DenseMatrix.zeros[Double](n,m)`	`c = zeros(n,m)`	`c = zeros((n,m))`
Zeroed vector	`val a = DenseVector.zeros[Double](n)`	`a = zeros(n)`	`a = zeros(n)`
Vector of ones	`val a = DenseVector.ones[Double](n)`	`a = ones(n)`	`a = ones(n)`
Vector of particular number	`val b = DenseVector.fill(n){5.0}`	`a = ones(n) * 5`	`a = ones(n) * 5`
Identity matrix	`DenseMatrix.eye[Double](n)`	`eye(n)`	`eye(n)`
Diagonal matrix	`diag(DenseVector(1.0,2.0,3.0))`	`diag([1 2 3])`	`diag((1,2,3))`
Matrix inline creation	`val a = DenseMatrix((1.0,2.0),(3.0,4.0))`	`a = [1 2; 3 4]`	`a = array([ [1,2], [3,4] ])`
Column vector inline creation	`val a = DenseVector(1,2,3,4)`	`a = [1 2 3 4]`	`a=array([1,2,3,4])`
Row vector inline creation	`val a = DenseVector(1,2,3,4).t`	`a = [1,2,3,4]'`	`a = array([1,2,3]).reshape(-1,1)`

Breeze methods

Indexing and Slicing

Operation	Breeze	Matlab	Numpy
Basic Indexing	`a(0,1)`	`a(1,2)`	`a[0,1]`
Extract subset of vector	`a(1 to 4)` or `a(1 until 5)` or `a.slice(1,4)`	`a(2:5)`	`a[1:4]`
(negative steps)	`a(5 to 0 by -1)`	`a(6:-1:1)`	`a[5:0:-1]`
(tail)	`a(1 to -1)`	`a(2:end)`	`a[1:]`
(last element)	`a( -1 )`	`a(end)`	`a[-1]`
Extract column of matrix	`a(::, 2)`	`a(:,3)`	`a[:,2]`

Breeze methods

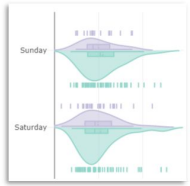
Operations

Operation	Breeze	Matlab	Numpy
Elementwise addition	`a + b`	`a + b`	`a + b`
Elementwise multiplication	`a :* b`	`a .* b`	`a * b`
Elementwise exponentiation	`a ^:^ b`	`a .^ b`	`power(a, b)`
Elementwise comparison	`a :< b`	`a < b` (gives matrix of 1/0 instead of true/false)	`a < b`
Elementwise equals	`a :== b`	`a == b` (gives matrix of 1/0 instead of true/false)	`a == b`
Inplace addition	`a :+= 1.0`	`a += 1`	`a += 1`
Inplace elementwise multiplication	`a :*= 2.0`	`a *= 2`	`a *= 2`
Vector dot product	`a dot b`, `a.t * b` [†]	`dot(a,b)`	`dot(a,b)`
Elementwise max	`max(a)`	`max(a)`	`a.max()`
Elementwise argmax	`argmax(a)`	`[v i] = max(a); i`	`a.argmax()`

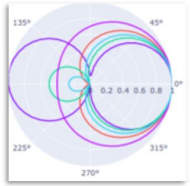
Breeze vis with Plotly

Fundamentals

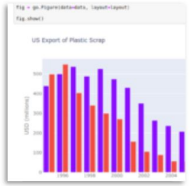
More Fundamentals »



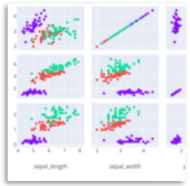
The Figure Data Structure



Creating and Updating Figures



Displaying Figures

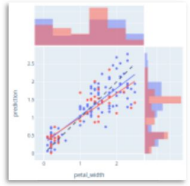


Plotly Express

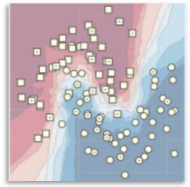
Analytical Apps with Dash

Artificial Intelligence and Machine Learning

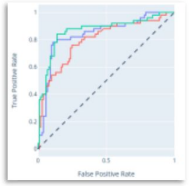
More AI and ML »



ML Regression



kNN Classification



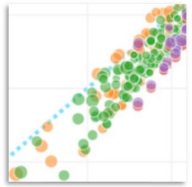
ROC and PR Curves

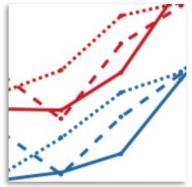
PCA Visualization

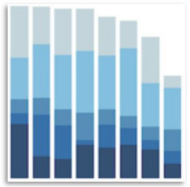
AI/ML Apps with Dash

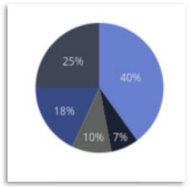
Basic Charts

More Basic Charts »









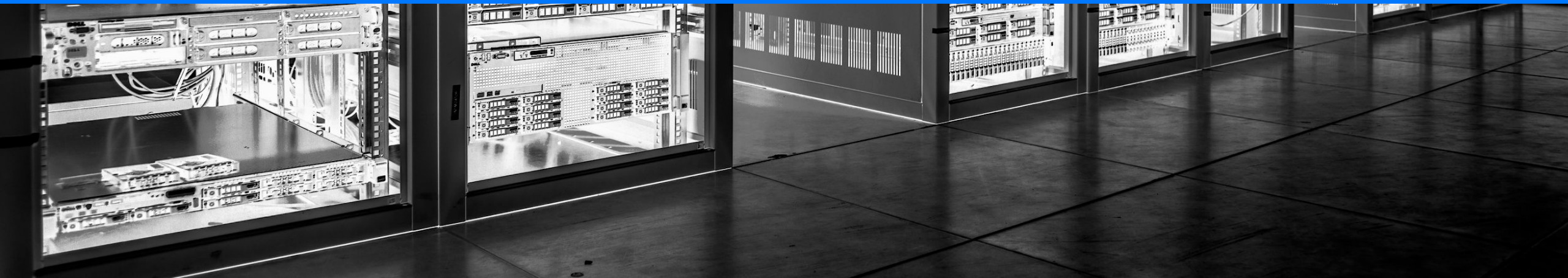
https://github.com/alexarchambault/plotly-scala

Almond (Jupyter) vs Zeppelin vs IDE

- Разрабатывать лучше сразу в IDE, так как там нет рандомных проблем с обертками, есть подсветка кода и удобство.
- Если хочется интерактивности и нативности, то Zeppelin
- Если хочется интерактивности и нет желания уходить с Jupyter, то Almond - Scala Kernel будет нормальным выходом https://almond.sh/
- Все инструменты просто ставятся через Docker

  Мой выбор IDEA + Big Data Tools plugin

Workshop



Lecture and workshop summary

1. Для полноценного погружения в Big Data нужно на базовом уровне владеть Java/Scala стеком
2. Для пользования Spark Scala является наиболее полной и быстрой опцией
3. Breeze - это Numpy для Scala
4. Scala и ее библиотеки в объеме DS это не страшно

Recommended links and literature

- Русскоязычное сообщество Scala https://t.me/scala_ru
- Scala Online REPL https://scastie.scala-lang.org/
- Хорошая онлайн-книга https://www.handsonscala.com/index.html
- Автор пишет про Scala и Spark https://alvinalexander.com/
- Упражнения по Scala https://www.scala-exercises.org/
- leetcode поддерживает Scala 🙂
- Хардкор и ФП со Scala https://underscore.io/books/scala-with-cats/

Module summary

1. Hadoop и его экосистема на сегодня это главная составляющая Big Data стека
2. Существует множество инструментов для работы с данными в HDFS начиная MapReduce и SQL, заканчивая Spark.
3. Самостоятельно развернуть базовую BigData инфраструктуру из готовых образов локально, либо с помощью инструментов облака не так сложно.
4. Знакомство с базовой Scala - хорошая инвестиция в дальнейший рост в области Big Data.

Спасибо за внимание