Talks & Presentations

Rodrigo Hernández Mota

2019-01-25

Contents

List of Tables

List of Figures

Preface

These are my talks and presentations in a nice & readable format.

How to read this book?

This is not a real book. Each chapter contains notes used as a reference guide to creates the slides.

About the author

to be defined.

Chapter 1

What is this?

1.1 The Short Answer

Q: What is this?

A: A site where I host and publish the documentation and reference materials for my talks.

1.2 The Long Answer

1.2.1 Motivation

Have you ever feel limited by the "expressiveness" of the documentation tools you use (e.g., simple markdown)? Or frustrated by the complexity they might introduce (e.g., latex)? Well, this situation inspired me to enter a quest to find the best technical writing tool for (software) engineers that's simple enough to learn, has a lightweight syntax, and avoids unnecessary complexity and boilerplate. I recently concluded that such a tool doesn't exist (but could!).

Since my background is on machine learning and data engineering, I was looking for something that can parse mathematical equations and at the same time, execute arbitrary code snippets. In particular, I was looking for:

- **A expressive but straightforward syntax for formatting.** We all know that LaTeX is the king for document formatting, but introduces significant boilerplate for small projects. On the other hand, markdown is very simple but compromises expressiveness.
- **Ability to parse and present math equations.** This is the main reason I keep coming back to LaTeX. Is there a way out?

- **Ability to execute arbitrary code snippets in several languages.** I don't ask for much. As an engineer specialized on the data fields, I expect to be able to run at least Scala, Python, and Bash.
- **Being able to use a single-source document to generate multiple outputs formats.** I might want to share the same document as a PDF, or a webpage.
- **Reference management simplification.** Nice to have!

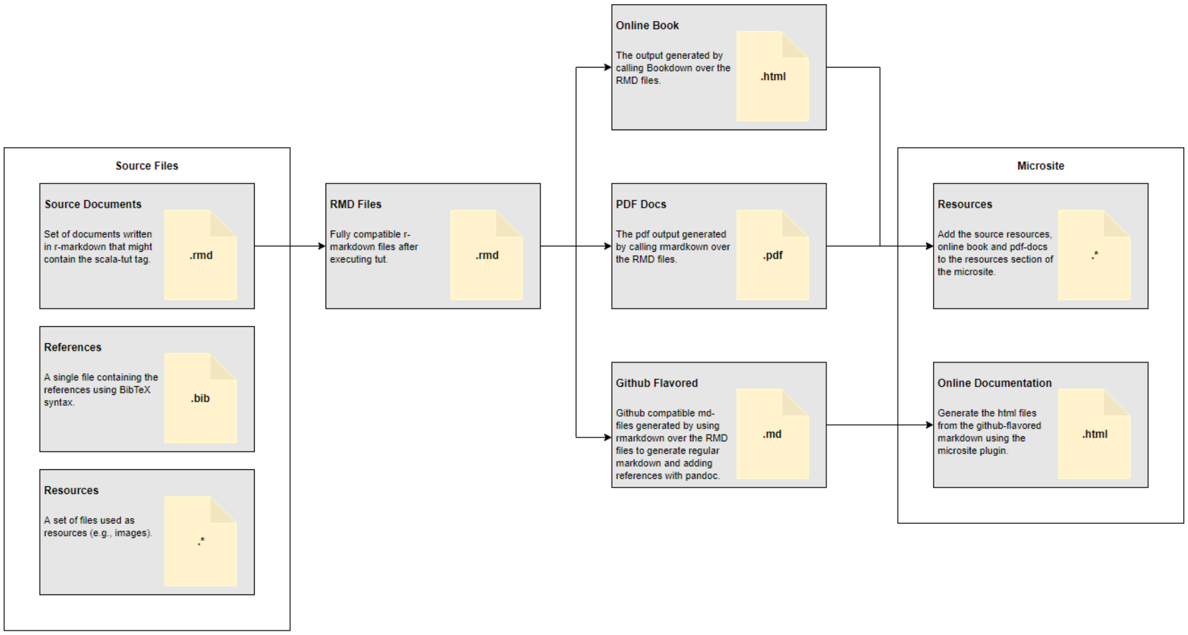
Fortunately, there are plenty of open source tools that aim to solve particular doc-oriented problems. My solution turned out to be a **giant hack** that combines several of these tools.

1.2.2 The proposed approach

You might be reading this document on a website, pdf-file, or even an ebook. A single source file generated all of these outputs. Let's be clear; this is not magic. It's the result of using several high-quality open source projects. In particular, I want to express my appreciation to all the collaborators of the following projects:

- Pandoc - a Haskell based universal document converter.
- Rmarkdown - a tool for reproducible research that allows computing code and narrative to be in the same document.
- Bookdown - an Rmarkdown extension.
- Scala Tut - an SBT plugin that allows executable scala-code snippets in Markdown.
- Scala Microsite - an SBT plugin that allows the creation of microsites.

Let me explain how to combine all these tools into a giant hack:


Source Files
Source Documents
Set of documents written in r-markdown that might contain the scala-tut tag.
.rmd
References
A single file containing the references using BibTeX syntax.
.bib
Resources
A set of files used as resources (e.g., images).
.*
RMD Files
Fully compatible r-markdown files after executing tut.
.rmd
Online Book
The output generated by calling Bookdown over the RMD files.
.html
PDF Docs
The pdf output generated by calling rmardkown over the RMD files.
.pdf
Github Flavored
Github compatible md-files generated by using rmarkdown over the RMD files to generate regular markdown and adding references with pandoc.
.md
Microsite
Resources
Add the source resources, online book and pdf-docs to the resources section of the microsite.
.*
Online Documentation
Generate the html files from the github-flavored markdown using the microsite plugin.
.html

Find a naive implementation of this process in the github repo that contains the source code for this site.

1.2.3 Why R markdown?

As you might have noticed, this Frankenstein tool relies mostly on R-markdown for the most relevant features. Let me cite the author of the Bookdown framework to explain my decision:

> "R Markdown may not be the right format for you if you find these elements not enough for your writing: paragraphs, (section) headers, block quotations, code blocks, (numbered and unnumbered) lists, horizontal rules, tables, inline formatting (emphasis, strikeout, superscripts, subscripts, verbatim, and small caps text), LaTeX math expressions, equations, links, images, footnotes, citations, theorems, proofs, and examples."

(Xie, Allaire, and Grolemund 2018)

1.2.4 Publishing

The SBT microsite plugin facilitates the deployment of the resulting site into github pages. This service can host and serve static sites without a problem by using Jekyll.

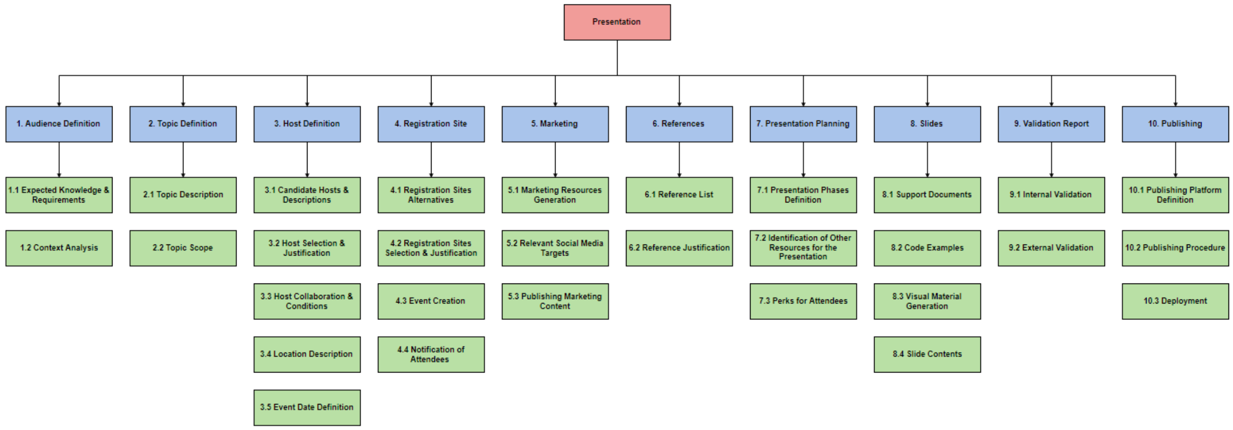
I use a custom script that execute the whole pipeline:

- `./publish --local`: serve the site in localhost.
- `./publush --site`: serve the site in github pages.

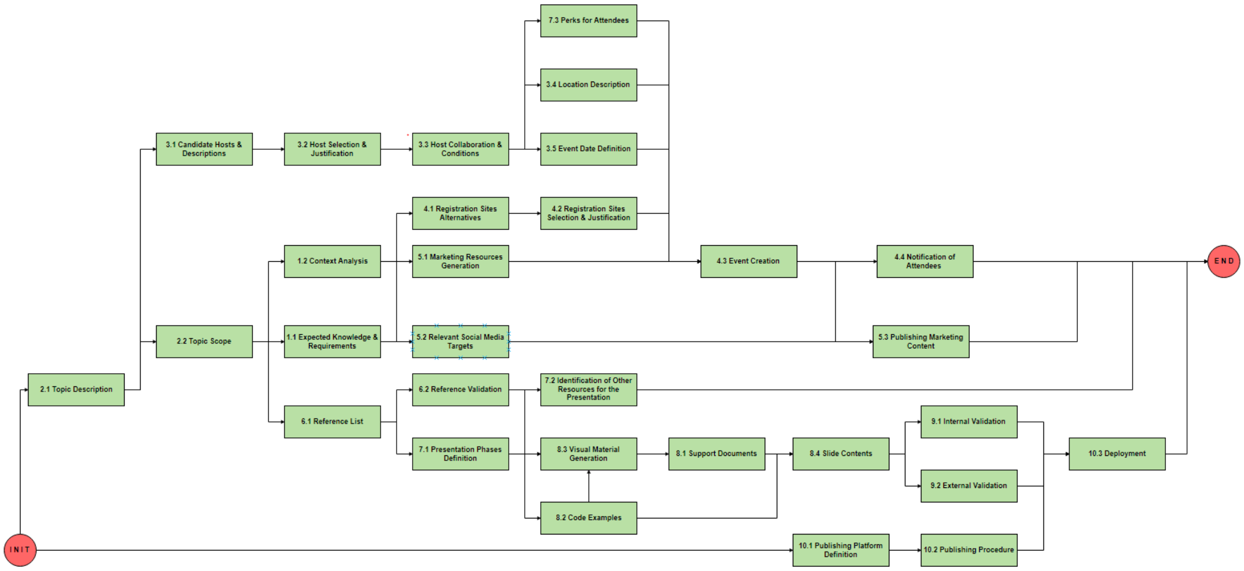
Note that you must run the `setup` scripts before.

1.2.5 Presentations

We can represent the project of generating a presentation with a work-breakdown structure diagram (WBS).


Presentation
1. Audience Definition
2. Topic Definition
3. Host Definition
4. Registration Site
5. Marketing
6. References
7. Presentation Planning
8. Slides
9. Validation Report
10. Publishing
1.1 Expected Knowledge & Requirements
1.2 Context Analysis
2.1 Topic Description
2.2 Topic Scope
3.1 Candidate Hosts & Descriptions
3.2 Host Selection & Justification
3.3 Host Collaboration & Conditions
3.4 Location Description
3.5 Event Date Definition
4.1 Registration Sites Alternatives
4.2 Registration Sites Selection & Justification
4.3 Event Creation
4.4 Notification of Attendees
5.1 Marketing Resources Generation
5.2 Relevant Social Media Targets
5.3 Publishing Marketing Content
6.1 Reference List
6.2 Reference Justification
7.1 Presentation Phases Definition
7.2 Identification of Other Resources for the Presentation
7.3 Perks for Attendees
8.1 Support Documents
8.2 Code Examples
8.3 Visual Material Generation
8.4 Slide Contents
9.1 Internal Validation
9.2 External Validation
10.1 Publishing Platform Definition
10.2 Publishing Procedure
10.3 Deployment

To represent the dependencies, we can take the "leaves of the tree" and arrange them in a network diagram.


7.3 Perks for Attendees
3.4 Location Description
3.1 Candidate Hosts & Descriptions
3.2 Host Selection & Justification
3.3 Host Collaboration & Conditions
3.5 Event Date Definition
4.1 Registration Sites Alternatives
4.2 Registration Sites Selection & Justification
1.2 Context Analysis
5.1 Marketing Resources Generation
4.3 Event Creation
4.4 Notification of Attendees
END
2.2 Topic Scope
1.1 Expected Knowledge & Requirements
5.2 Relevant Social Media Targets
5.3 Publishing Marketing Content
2.1 Topic Description
6.2 Reference Validation
7.2 Identification of Other Resources for the Presentation
6.1 Reference List
9.1 Internal Validation
7.1 Presentation Phases Definition
8.3 Visual Material Generation
8.1 Support Documents
8.4 Slide Contents
10.3 Deployment
9.2 External Validation
8.2 Code Examples
INIT
10.1 Publishing Platform Definition
10.2 Publishing Procedure

1.3 References

Chapter 2

Monads in [My Py]thon

2.1 Acknowledgments

This talk is inspired and based on the following conferences:

- Monads, in my Python? by Xuanyi Chew
- Scala Monads: Declutter Your Code With Monadic Design by Dan Rosen
- Category Theory, The essence of interface-based design by Erik Meijer
- Type-checked Python in the real world by Carl Meyer
- Learning to Love Type Systems by Lauren Tan
- New Functional Constructs in Scala 3 by Martin Odersky

2.2 About MyPy

The *python-enhancement-proposal-484* (PEP 484) by Guido van Rossum and Jukka Lehtosalo introduced the concept of **type hints** inspired on function annotations (PEP 3107). This type-hints are completely ignored at runtime but can be used with an optional static type checker. MyPy is the most popular type checker for python, lead by Guido van Rossum at Dropbox.

2.2.1 Why types

> "A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute."

(Pierce and Benjamin 2002)

> "A type system is a way to add constraints to your code. The type system is there to help you enforce logic that's consistent within those constraints."

(Tan 2018)

Constraints are desirable because they limit the number of bugs in the program. We can use a strong DSL (domain specific language) to represent the business logic of our application and let the type checker verify the consistency.

In the Pragmatic types blogpost, the author explains the difference between using a type system for type-checking the code vs using unit-tests. Consider the following illustration:

We achieve type-safety in an application with (1) a robust type system & checker, and (2) by following the functional programming principles.

Functional programming started as a practical implementation of the following mathematical theories:

- **Proof Theory**: logic and mathematical proofs.
- **Category Theory**: algebra, composition and abstractions.
- **Type Theory**: programs and the idea of prepositions as types.

Curry-Howard-Lambek correspondence shows that these three theories are equivalent among each others.

Consider the following python function:

```python
def addition(x: int, y: int) -> int:     # proposition
    return x + y                         # proof
```

The type signature serves as a proposition; given two integers `x` and `y`, there exists a function that returns another integer.

The implementation (body) of the function is the proof of such proposition. In this sense, **types** are propositions and **programs** are proofs. Therefore, we can think of type-checking as proof-checking.

Good type signatures and a DSLs facilitate the implementation of a particular program and let's the developer rely on the type-systems to increase productivity.

2.2.2 Installation

Installation it's straightforward (Ubuntu 18.04):

```
$ sudo apt install python3.7 && python3.7 -m pip install -U mypy
```

Now you can run the static type checker with your python programs:

```
$ python3.7 -m mypy app.py
```

To avoid warnings/errors related to external libraries, use:

```
$ python3.7 -m mypy --ignore-missing-imports app.py
```

2.3 About Monads

The most popular definition of a monad is probably the one phrased by James Iry in his blog-post A Brief, Incomplete, and Mostly Wrong History of Programming Languages.

> "A monad is just a monoid in the category of endofunctors."

Nonetheless, we can find the complete form of this definition in the book Categories for the working mathematician.

> "A monad in X is just a monoid in the category of endofunctors of X, with product × replaced by composition of endofunctors and unit set by the identity endofunctor."

(Mac Lane 2013)

And a more formal definition in this same book:

> "Formally, the definition of a monad is like that of a monoid M in sets. The set M of elements of the monoid is replaced by the endofunctor $T : X \to X$, while the cartesian product × of two sets is replaced by the composite of two functors, the binary operation $\mu : M \times M \to M$ of multiplication by the trasformation $\mu : T^2 \to T$ and the unit (identity) element $\nu : 1 \to M$ by $\nu : I_x \to T$."

(Mac Lane 2013)

$\mu : T^2 \to T$

With the help of this stackoverflow post, this wolfram post and the scala cats typelevel docs we can shine some light to this definition:

- A monoid is a representation of a set S closed under an associative binary operation and has an identity element or unit.

A type `A` can form a semigroup if it has an associative binary operation `combine` that satisfies `combine(x, combine(y, z)) = combine(combine(x, y), z)` for any choice of `x`, `y`, and `z` in `A`.

```
trait Semigroup[A] {
    def combine(x: A, y: A): A
}

object Semigroup {
    def combine[A](x: A, y: A)(implicit sg: Semigroup[A]): A =
        sg.combine(x, y)
}
```

We can create a simple example for `Int`:

```
implicit val integerAdditionSemigroup: Semigroup[Int] =
    new Semigroup[Int] {
        def combine(x: Int, y: Int): Int = x + y
    }
```

Example:

```
Semigroup.combine[Int](1, 2)
// res0: Int = 3

Semigroup.combine[Int](1, Semigroup.combine[Int](2, 3))
// res1: Int = 6
```

To define a monoid we need to extend the `Semigroup` with an empty value such that the following holds true: `combine(x, empty) = combine(empty, x) = x`

```
trait Monoid[A] extends Semigroup[A] {
  def empty: A
}

object Monoid {
    def empty[A](implicit m: Monoid[A]): A = m.empty
    def combine[A](x: A, y: A)(implicit m: Monoid[A]): A =
        m.combine(x, y)
}

// Int monoid
implicit val integerAdditionMonoid: Monoid[Int] = new Monoid[Int] {
    def empty: Int = 0
    def combine(x: Int, y: Int): Int = x + y
}
```

We can verify the `combine` operation with our empty element:

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[loop above, "1_A"] \arrow[dr, "g \circ f"'] & B \arrow[d, "g"] \arrow[loop right, "1_B"] \\
 & C \arrow[loop right, "1_C"]
\end{tikzcd}$$

Figure 2.1: Categories

```
Monoid.combine[Int](1, Monoid.empty[Int])
// res3: Int = 1
```

- A **functor** is a mathematical structure-preserving transformation between categories. And **endofunctor** is a functor from one category back to the same category.

```
trait Functor[F[_]] {
  def map[A, B](fa: F[A])(f: A => B): F[B]
}
```

- A **category** is a collection of (1) objects, (2) morphisms or arrows for each pair of objects, and a (3) binary operation for composition between arrows. See more about categories.

According to Erik Meijer in this talk "Category Theory, The essence of interface-based design" we can use the following following equivalences as a practival guide:

- `Category` = Programming Language
- `Objects` = Types
- `Morphism` = functions, static methods, properties : `f(a: A): B` or `f: B` an `A`

(Meijer 2015)

2.4 Monads in Scala

The Scala language provides a rich set of functional programming constructs. Consider the following code-snipped shown at the conference “Scale by the Bay - 2018” by Martin Odersky to define an abstract monad in Scala:

```
trait Functor[F[_]] {
    def map[A, B](this x: F[A])(f: A => B): F[B]
}

trait Monad[F[_]] extends Functor[F] {
    def pure[A](x: A): F[A]
    def flatMap[A, B](this x: F[A])(f: A => F[B]): F[B]
    def map[A, B](this x: F[A])(f: A => B): F[B] =
        x.flatMap(f `andThen` pure)
}
```

Now we can use extension methods (Scala 3) to create a particular implementation:

```
implicit object ListMonad extends Monad[List] {
    def flatMap[A, B](this xs: List[A])(f: A => List[B]): List[B] =
        xs.flatMap(f)
    def pure[A](x: A): List[A] = List(x)
}
```

(Odersky 2018)

2.5 Monads in python

Without higher-kinded types. For now.

Consider the following python functions:

```

def div(num: int, den: int) -> int:
    return num / den

def factorial(n: int) -> int:
    if n < 0:
        raise Exception("Factorial is defined over non-negative numbers")
   return 1 if n == 0 else n * factorial(n-1)
```

If we would like to compose both functions we would likely have to implement several safe guards to avoid runtime erros and invalid inputs. What if we use python’s `None` naively instead of error-handling for the `div` function?

```python
def div(num: int, den: int) -> int:
    if den == 0:
        return None
    return num / den
```

We still have composability problems (see this diagram). Moreover, our types are incorrect!

- **Q**: Is there a way we can generalize this?
- **A**: Monads!

Let's create an `Option` monad.

For simplicity, let's use a higher-order function that allows us to compose two functions:

```python
from typing import Callable, TypeVar

A = TypeVar('A')
B = TypeVar('B')
C = TypeVar('C')
```

```python
def compose(this: Callable[[A], B], and_then: Callable[[B], C]) -> Callable[[A], C]:
    return lambda x: and_then(this(x))
```

Now let's define our option:

```python
from abc import ABC, abstractmethod
from typing import Union, Generic, TypeVar, Callable

A = TypeVar("A", covariant=True)
B = TypeVar("B")
T = TypeVar("T")
```

```python
class Option(Generic[A], ABC):

    @abstractmethod
    def __str__(self) -> str:
        pass

    @abstractmethod
    def get(self, or_else: B) -> Union[A, B]:
        pass
    @abstractmethod
    def flat_map(self, f: Callable[[A], 'Option[B]']) -> 'Option[B]':
        pass

    @staticmethod
    def pure(x: T) -> 'Option[T]':
        return Some(x)

    def map(self, f: Callable[[A], B]) -> 'Option[B]':
        return self.flat_map(compose(this=f, and_then=self.pure))

    @abstractmethod
    def foreach(self, f: Callable[[A], None]) -> None:
        pass

    @abstractmethod
    def flatten(self) -> 'Option':
        pass
```

An `Option[A]` can take `Some[A]` value or be `Empty`. We can define the `Some` type:

```
class Some(Option[A]):
    def __init__(self, value: A) -> None:
        self._value = value

    def __str__(self) -> str:
        return f"Some({self._value})"

    def get(self, or_else: B) -> Union[A, B]:
        return self._value

    def flat_map(self, f: Callable[[A], Option[B]]) -> Option[B]:
        return f(self._value)

    def foreach(self, f: Callable[[A], None]) -> None:
        f(self._value)

    def flatten(self) -> Option:
        if isinstance(self._value, Option):
            return self._value.flatten()
        return self
```

The `Empty` class is defined as:

```
class Empty(Option[A]):
    def __init__(self) -> None:
        pass

    def __str__(self) -> str:
        return "Empty"

    def get(self, or_else: B) -> Union[A, B]:
        if isinstance(or_else, Exception):
            raise or_else
        return or_else

    def flat_map(self, f: Callable[[A], Option[B]]) -> Option[B]:
        return Empty[B]()

    def foreach(self, f: Callable[[A], None]) -> None:
        return None

    def flatten(self) -> Option:
        return self
```

Now we can use our option type!

```
# Two options
opt_a: Option[int] = Some(2)
opt_b: Option[int] = Some(5)

# Sum a+b
opt_c = opt_a.flat_map(lambda a: opt_b.map(lambda b: a + b))

# Sum c+d
opt_d: Option[int] = Empty()
opt_e = opt_c.flat_map(lambda c: opt_d.map(lambda d: c + d))

# Print results
print(f"opt_c = {opt_c}\nopt_e = {opt_e}")
```

Let's define some decorators:

```
from typing import Callable, TypeVar

T = TypeVar("T")
A = TypeVar("A")
```

Decorate a function to output `Option` type:

```
def to_option(fn: Callable[..., T]) -> Callable[..., Option[T]]:
    def inner(*args, **kwargs) -> Option[T]:
        try:
            value = fn(*args, **kwargs)
            if value is None:
                return Empty[T]()
            return Some(value)
        except Exception:
            return Empty[T]()
    return inner
```

Decorate a function facilitate `Option` composability;

```
def composable(fn: Callable[..., Option[T]]) -> Callable[..., Option[T]]:
    def inner(*args, **kwargs) -> Option[T]:
        new_args = []
        new_kwargs = {}
        for arg in args:
            new_arg = arg if isinstance(arg, Option) else Some(arg)
            new_arg.foreach(lambda value: new_args.append(value))
        for k in kwargs:
            v = kwargs[k]
            new_val = v if isinstance(v, Option) else Some(v)
            new_val.foreach(lambda value: new_kwargs.update({k: value}))
        return fn(*new_args, **new_kwargs)
    return inner
```

Now we are ready to define our functions:

```
@composable
@to_option
def div(num: int, den: int) -> int:
    return num / den
```

```
@composable
@to_option
def factorial(n: int) -> int:
    if n < 0:
        raise Exception("Factorial is defined over non-negative numbers")
    return 1 if n == 0 else n * factorial(n-1)
```

Our monadic values allows us to easily compose between `Objects` (see this).

```
a = 5
b = 0
res = div(a, b)
print(f"div(a,b) = {res}")
```

```
a = 15
b = 0
c = 3
d = 5
res_1 = div(d, div(a, b))
res_2 = div(d, div(a, c))
print(f"div(d, div(a, b) = {res_1}\ndiv(d, div(a, c)) = {res_2}")
```

```
a = 10
b = -2
res_1 = div(a, b)
res_2 = factorial(res_1)
print(f"div(a, b) = {res_1}\nfactorial(res_1)= {res_2}")
```

Great!

2.6 Example

add a more complex example.

2.7 References

Chapter 3

End-to-End ML with Apache Spark

3.1 Outline

1. ML Project Overview
2. Operationalizing
3. Spark-Based Projects
4. Code Example!

3.2 Starting Questions

- Who in here is a Data Professional (e.g., scientist, engineer)?
- How many of you have used Apache Spark? in production?
- How many of you are currently developing/maintaining a ML service?

3.3 Acknowledgments

This talk is based and inspired on the following conferences:

- Operationalizing Machine Learning - Serving ML Models by Boris Lublinsky
- Concept Drift: Monitoring Model Quality in Streaming Machine Learning Applications by Emre Velipasaoglu
- R, Scikit-Learn, and Apache Spark ML: What Difference Does It Make? by Villu Ruusmann

Find the complete list of references in the **References** section.

3.4 Prerequisites

This talk assumes you are a machine learning enthusiast or a data-professional (e.g. scientist, engineer) that is well aware the basic concepts required to design and execute an ML-project.

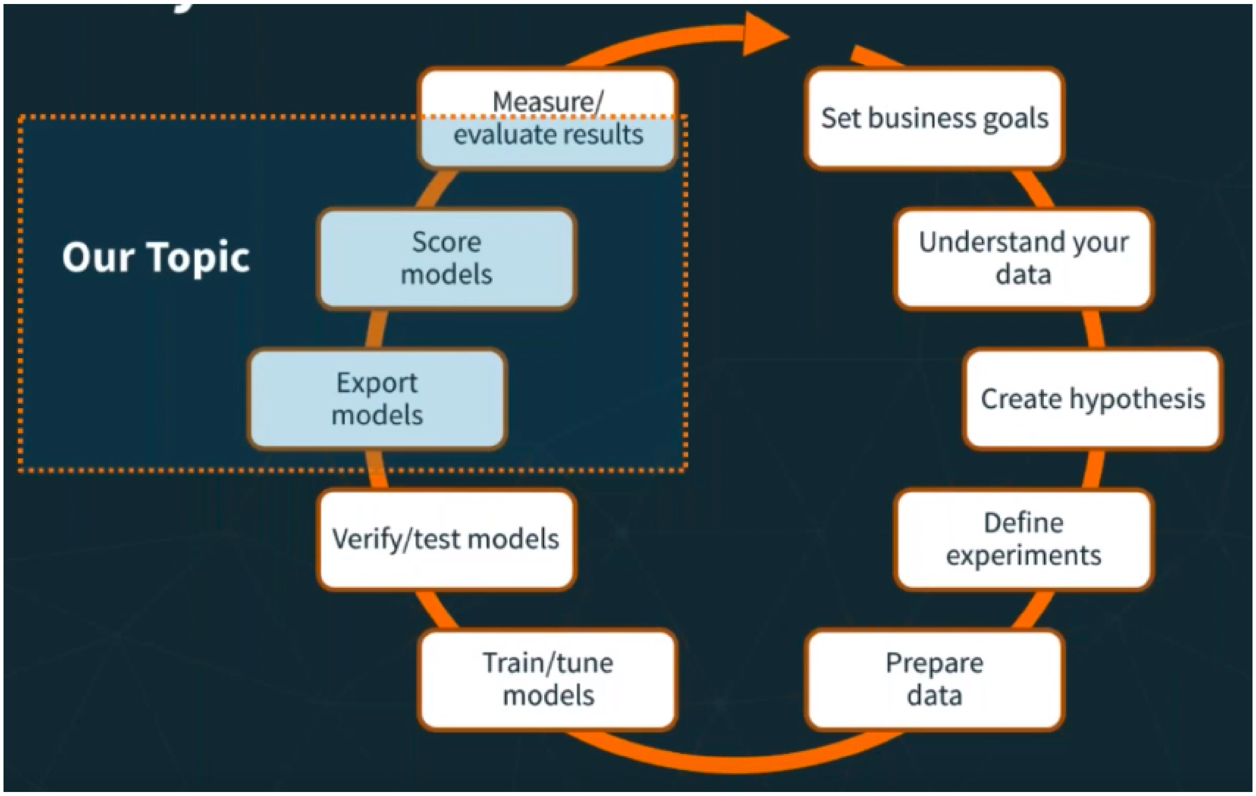
The audience must have a workable understanding of:

- Main programming languages used in the data-sphere (i.e. scala, python, R)
- General understanding of data architecutres (e.g. batch-oriented, streaming)
- Machine Learning theory (i.e., lots of math).
- Machine Learning frameworks (e.g., Spark ML, Tensorflow, PyTorch)
- Data-related skills (e.g., cleaning, visualization)

3.5 ML Project Overview

Typically with a ML Project, different groups are responsible for model training and serving. Moreover, the data science toolbox is constantly evolving, pushing software engineers to create more model-serving frameworks and introducing complexity to the development pipeline.

Consider the following machine-learning pipeline:



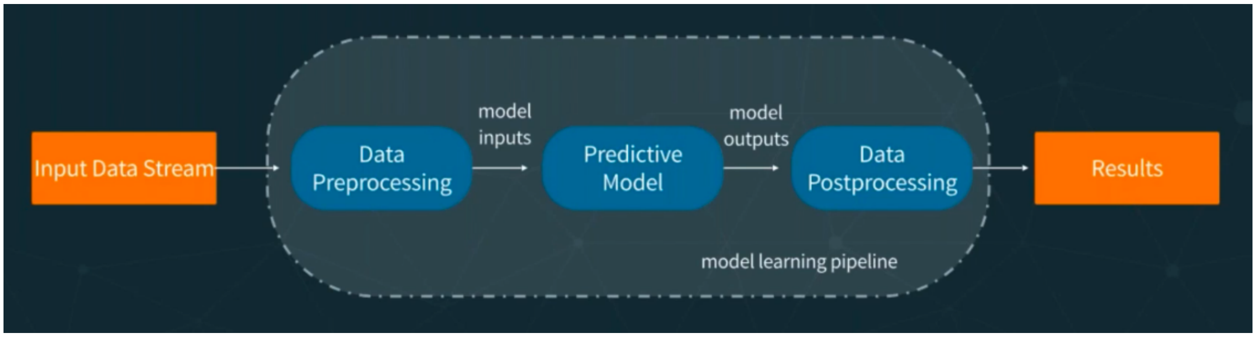
Machine Learning Cycle.

3.5.1 What's a ML Model?

We will use the idea of a model as just a function `f` that transforms a set of inputs `x` into outputs `y` (i.e. `y = f(x)`).

This definition allows us to apply functional composition in the implementation of our ML service.

With this is mind, we can introduce the concept [machine learning pipelines] as a graph defining a chain of operations (e.g., data transformations):

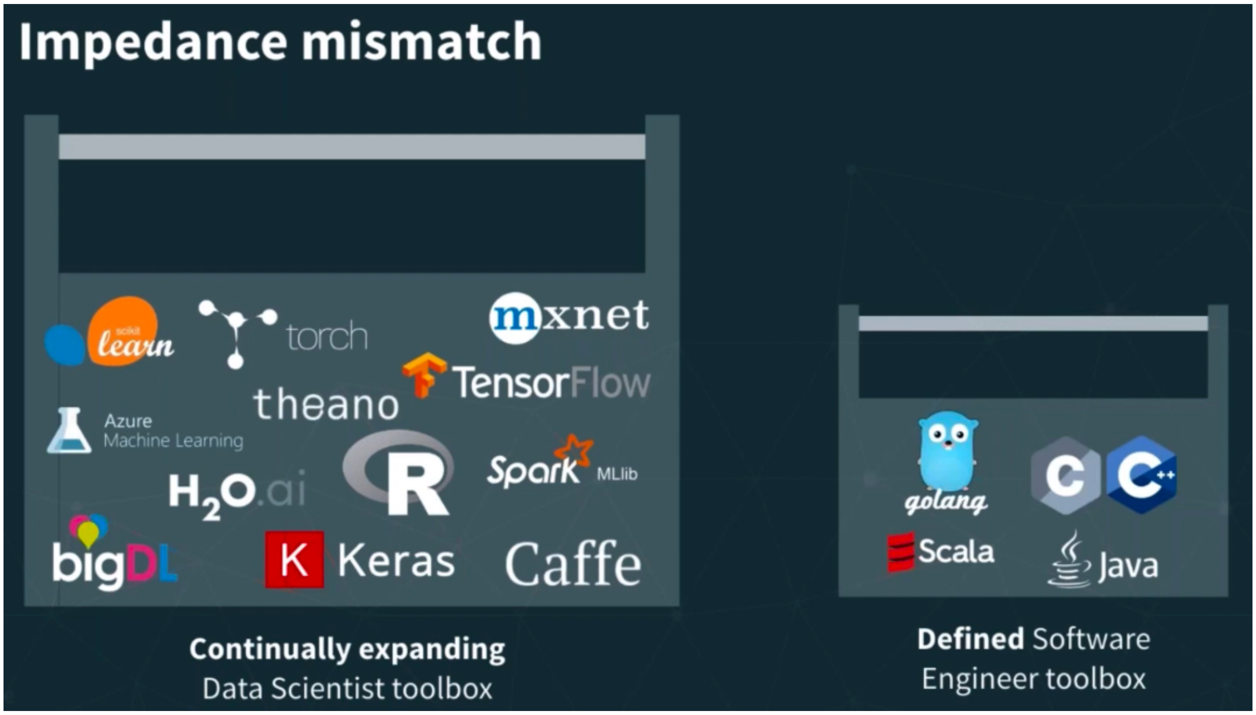


Why is it important to define a pipeline? To encapsulate all the logic needed to serve the machine learning model. This formalizes the pipeline form the input data to the output.

3.6 Operationalizing

3.6.1 Traditional Approach

Traditionally, the machine learning model was viewed as code. This code had to be somehow imported for serving in production.



Impedance mismatch

Continually expanding Data Scientist toolbox

Defined Software Engineer toolbox

Impedance mismatch!

3.6.2 A simple solution

We can shift our thinking from a "code" perspective to a "data" perspective and represent the model using a standard specification that's agnostic to the training process. We can use the PMML specification designed by the Data Mining Group to achieve this.

Predictive Markdown Model Language is:

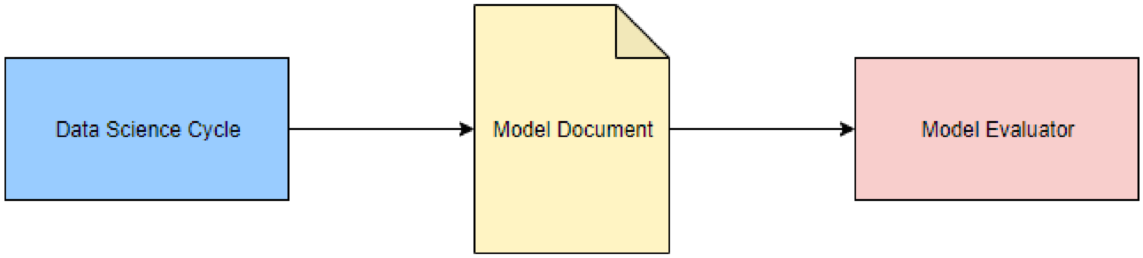
> "an XML-based language that provides a way for applications to define statistical and data-mining models as well as to share models between PMML-compliant applications."

(Ruusmann 2017)

Integration with the most popular ML frameworks via JPMML:

- jpmml-sparkml
- jpmml-sklearn
- jpmml-r
- jpmml-xgboost
- jpmml-tensorflow

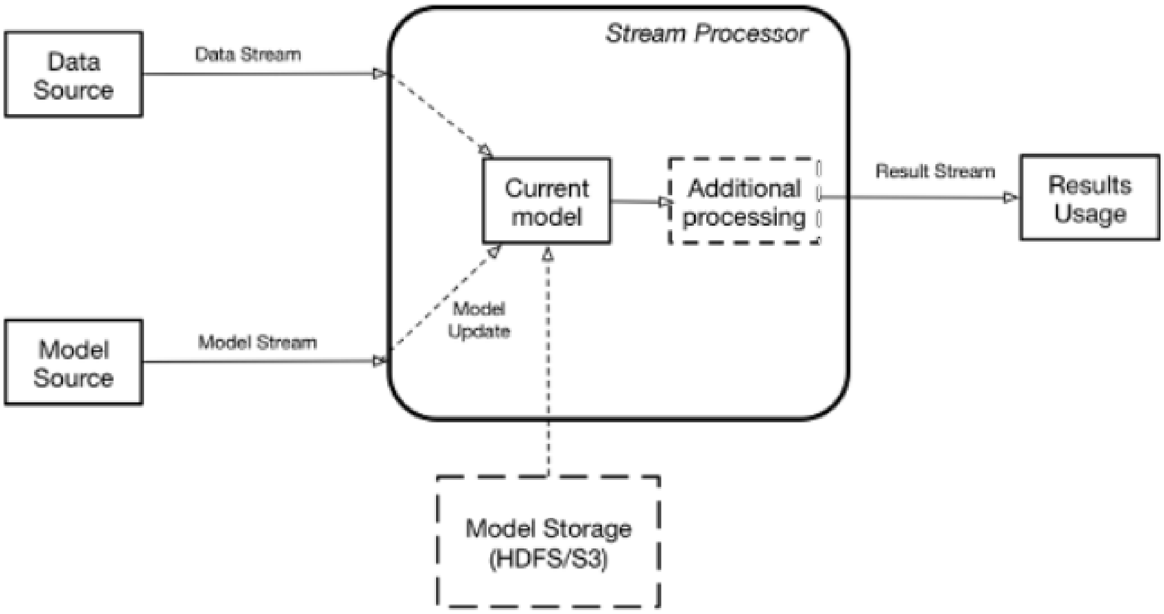
Using these tools we can achieve:



Simple Scoring.

3.6.3 Best Practice

We can use either a stream-processing engine (SPE e.g., Apache Spark, Flink) or a stream-processing library (SPL e.g., Akka Stream, Kafka Stream).



Suggested architecture.

- SPE: Good fit for applications that require features provided out of the box by such engines.
- SPL: Provide a programming model highly customizable and light-weight.

(Lublinsky 2017)

We can use Akka Streams - based on Akka Actors, to implement the proposed architecture (see syntax example). The result would look like this:

Simple Akka Implementation

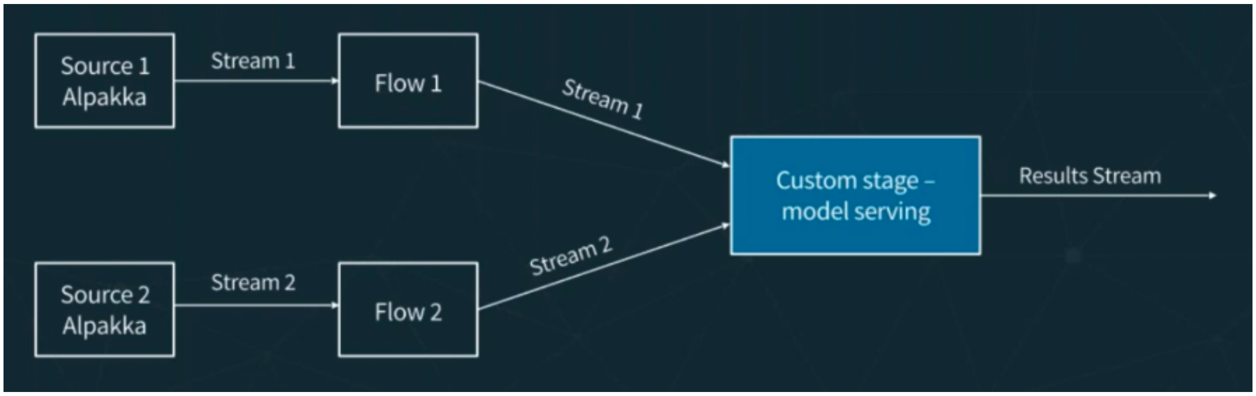


Figure 3.1: Naive Akka Implementation

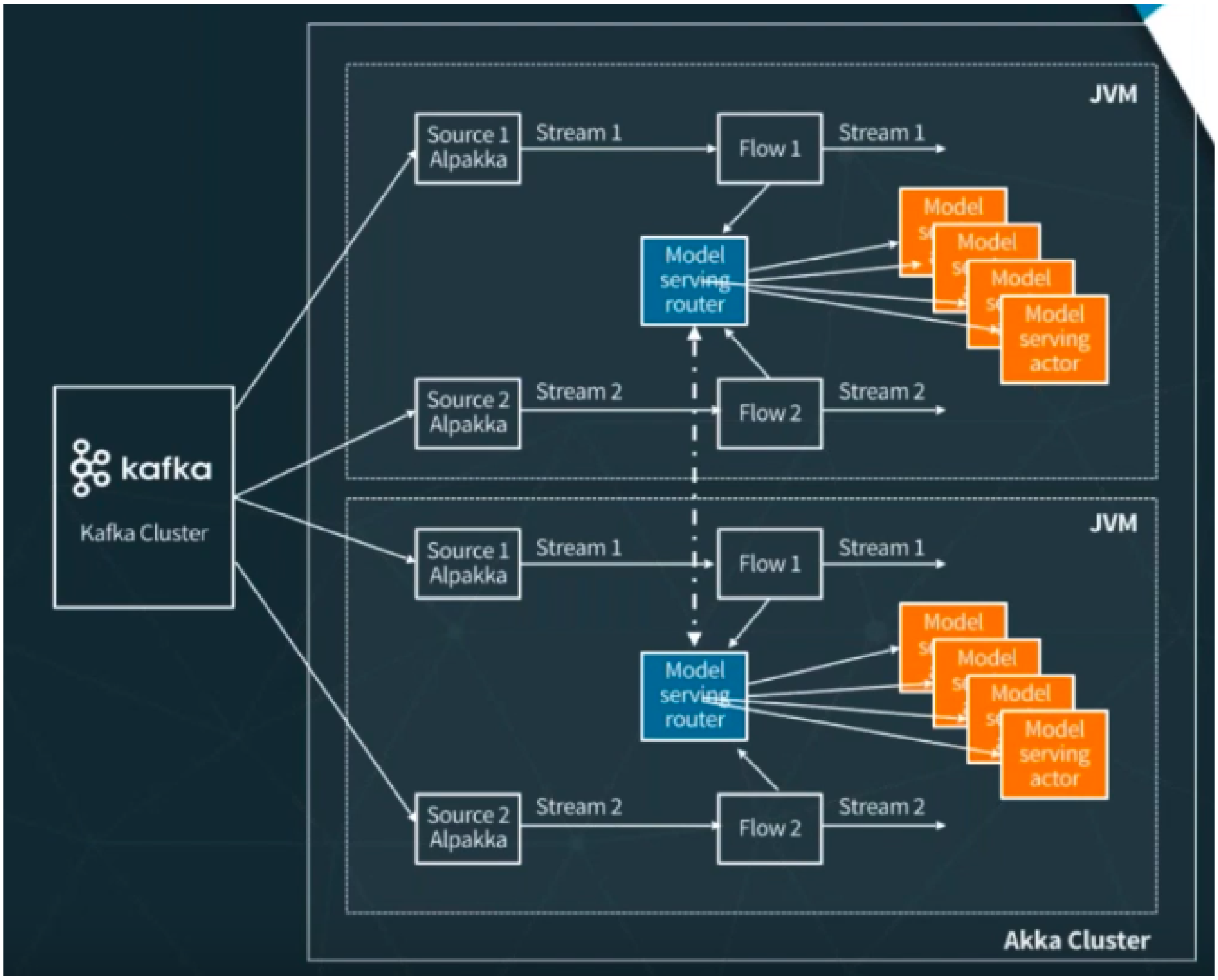


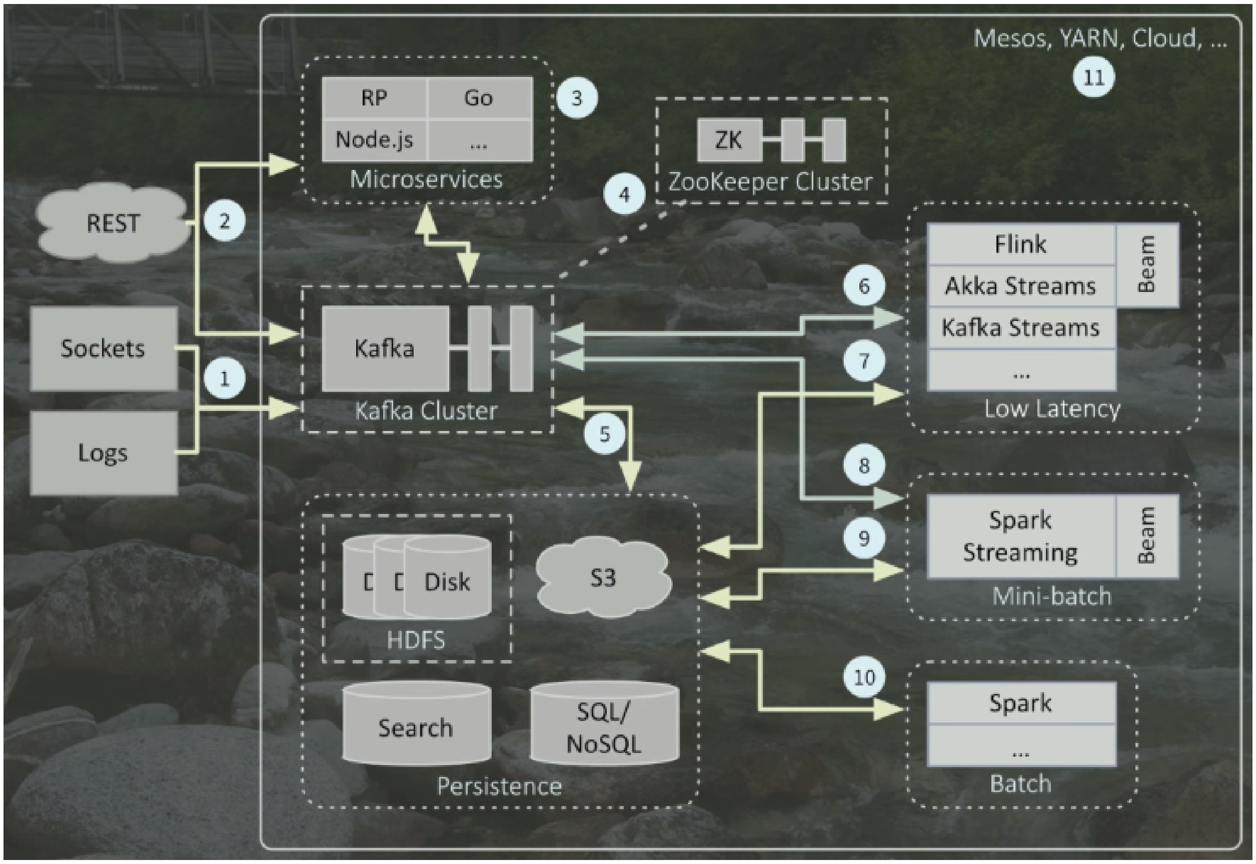
Figure 3.2: Akka Cluster Implementation

Furthermore, we can enhance this approach by using Akka Clusters.

Akka Cluster Implementation

3.6.4 The Big Picture

Dean Wampler does a fantastic job describing the overall picture of a data-driven system architecture.



Big Picture Architecture

(Wampler 2017)

3.7 Spark-Based Projects

3.7.1 Why Apache Spark?

According to their website,

> "Apache Spark is a unified analytics engine for large-scale data processing."

According to the book “High Performance Spark - Best Practices for Scaling & Optimizing Apache Spark”:

> “Apache Spark is a high-performance, general puropose distributed computer system. Spark enables us to process large quantities of data, beyond what can fit on a sinlge machine, with a high-level, relatively easy-to-use API. Uniquely, Spark allows us to write the logic of data transformations and machine learning algorithms in a way that is parallelizable, but relatively system agnostic.”

(Karau and Warren 2017)

Most of the Apache Spark features revolve around a base data-structure called RDDs (resilient distributed datasets). An RDD is a fault-tolerant collection of elements that can be operated on parallel.

Let’s initialize an Spark Session (sbt console: `sbt -Dscala.color "content/console"`):

```
import org.apache.spark.sql.SparkSession

val spark =
  SparkSession.builder.appName("Example!").config("spark.master", "local[*]").getOrCreat

import spark.implicits._
```

By default, the number of partitions is the number of all available cores (Laskowski 2017):

```
spark.sparkContext.defaultParallelism
// res0: Int = 12
```

We can test this by creating a simple Dataset from a list:

```

trait Person

object Person {
  final case class Dead(name: String, birthYear: Int, deadYear: Int) extends Person {
      def kill: Dead = this
    }

  final case class Alive(name: String, birthYear: Int) extends Person {
    def kill: Dead = Dead(name, birthYear, 2019)
  }

  val names: List[String] = List(
    "Data Ninja",
    "Random Developer",
    "Pizza Lover",
    "Beer Lover"
  )

  val years: List[Int] = (1980 to 2000).toList

  def getRandomElement[A](ls: List[A]): A =
    ls(scala.util.Random.nextInt(ls.size))

  def getRandom: Alive = Alive(getRandomElement(names), getRandomElement(years))
}

val people: List[Person.Alive] = (1 to 1000).toList.map(i => Person.getRandom)
```

We can now create a `Dataset[Person]`:

```
import org.apache.spark.sql.Dataset

val alivePeople: Dataset[Person.Alive] = spark.createDataset(people)
```

The number of partitions on this dataset:

```
alivePeople.rdd.partitions.size
// res1: Int = 12
```

```
val deadPeople: Dataset[Person.Dead] =
  alivePeople.filter(_.birthYear > 1994).map(person => person.kill)
// deadPeople: org.apache.spark.sql.Dataset[Person.Dead] = [name: string, birthYear: i

deadPeople.show()
// +----------------+---------+--------+
// |            name|birthYear|deadYear|
// +----------------+---------+--------+
// |      Data Ninja|     1998|    2019|
// |Random Developer|     2000|    2019|
// |Random Developer|     1996|    2019|
// |      Data Ninja|     1995|    2019|
// |Random Developer|     1996|    2019|
// |      Beer Lover|     1997|    2019|
// |      Beer Lover|     1997|    2019|
// |     Pizza Lover|     1997|    2019|
// |      Data Ninja|     1999|    2019|
// |      Beer Lover|     1997|    2019|
// |      Data Ninja|     1996|    2019|
// |      Beer Lover|     1996|    2019|
// |      Beer Lover|     1997|    2019|
// |     Pizza Lover|     1997|    2019|
// |      Data Ninja|     1995|    2019|
// |      Beer Lover|     1999|    2019|
// |Random Developer|     1995|    2019|
// |     Pizza Lover|     1995|    2019|
// |      Data Ninja|     1997|    2019|
// |     Pizza Lover|     2000|    2019|
// +----------------+---------+--------+
// only showing top 20 rows
//
```

```
spark.close()
```

For performance reasons, this presentation will use the official Scala API.

3.7.2 Intro to Spark ML

Spark ML is a practical and scalable machine learning library based on a [Dataset]. A Dataset is a distributed collection of data with interesting features such as strong typing, lambda functions, and with the advantages of the Spark SQL's optimized execution engine. We can manipulate a dataset with functional transformantions. The most basic ones:

- map - `Dataset[A].map(fn: A => B): Dataset[B]`
- flatMap - `Dataset[A].flatMap(fn: A => Dataset[B]): Dataset[B]`
- filter - `Dataset[A].filter(fn: A => Boolean): Dataset[A]`

One of the most usefull abstractions available on the Spark ML package are pipelines. Main concepts:

- `Dataset[Row]`: A set of data, also called dataframe. Each row usually represents an observation.
- `Transformer`: an algorithm that takes one `DataFrame` and returns another `DataFrame`.
- `Estimator`: an algorithm that takes a `DataFrame` and returns a `Transformer`.
- `Pipeline`: a chain of multiple `Transformer` or `Estimator`.

3.7.3 Intro to JPMML and Openscoring

Data Scientist might use Python and R for exploration and modeling while software engineers use Scala, Java, or Go for the system architecture. Complexity arises when dealing with multiple runtimes and trying to integrate the data solutions into the system. One way to standardize this interaction is via PMML: Predictive Markdown Model Language.

To use the jpmml-sparkml library, just add the following dependency to your sbt file:

```
"org.jpmml" % "jpmml-sparkml" % "1.4.5"
```

Now we can just take a Spark `PipelineModel` and create a PMML object:

```
val pmmlBuilder = new PMMLBuilder(schema, pipelineModel)
pmmlBuilder.build()
```

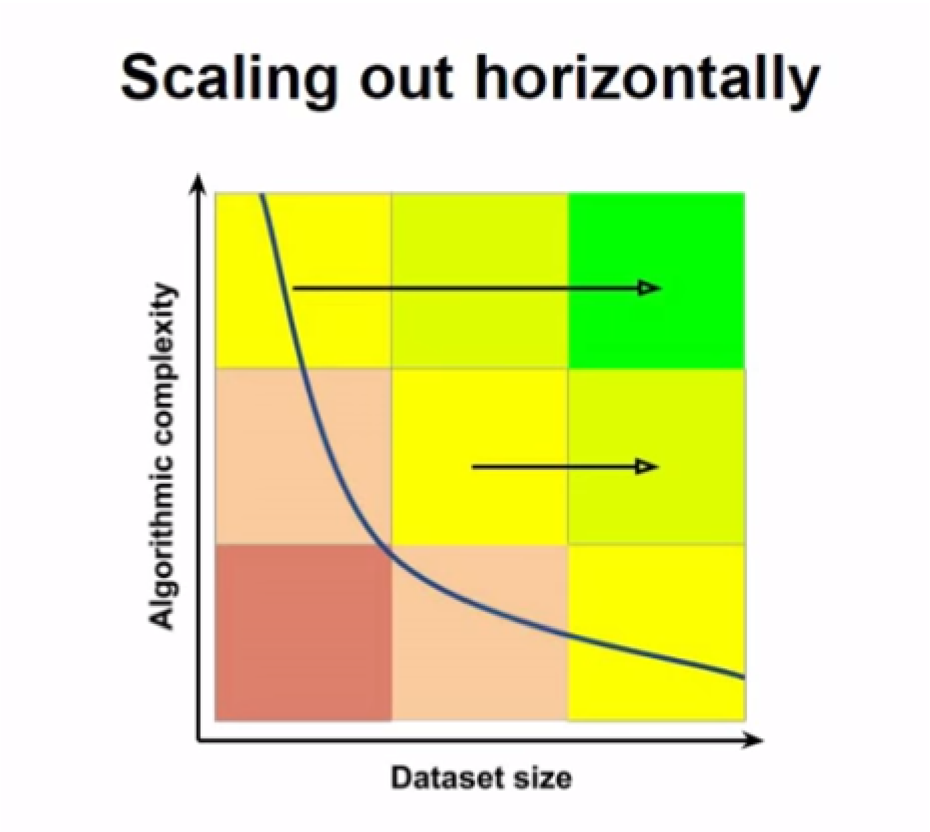
See the official jpmml-sparkml github repo for a complete list of supported `PipelineStages` types.

We can use Openscoring, a java-based REST web-service, as our scoring-engine of the resulting PMML model.

- Simple but powerful API
- Allows for single predictions and for batch predictions.
- Acceptable performance (usually sub-milliseconds respond time)

Model REST API endpoints:

HTTP method	Endpoint	Required role(s)	Description
GET	/model	-	Get the summaries of all models
PUT	/model/${id}	admin	Deploy a model
GET	/model/${id}	-	Get the summary of a model
GET	/model/${id}/pmml	admin	Download a model as a PMML document
POST	/model/${id}	-	Evaluate data in “single prediction” mode
POST	/model/${id}/batch	-	Evaluate data in “batch prediction” mode
POST	/model/${id}/csv	-	Evaluate data in “CSV prediction” mode
DELETE	/model/${id}	admin	Undeploy a model



Complexity vs dataset size.

(Ruusmann 2017)

3.8 Code Example!

3.8.1 Download the data

We can use the Gutenberg Project as a data-source for our ML task. To download the complete content of the gutenberg project as a set of txt-files run the following bash-command:

```
curl -sSL https://raw.githubusercontent.com/RHDZMOTA/spark-wordcount/develop/gutenberg.s
```

Depending on your network speed this can take up to 3 hours.

Let's figure out the "footprint" of this dataset:

- Number of books: `ls -l gutenberg | wc -l`
- Data size: `du -sh gutenberg`

Consider taking a random sample to facilitate local development. The following command generates a sample of 5K books:

```
mkdir gutenberg-sample && ls gutenberg/ | shuf -n 5000 | xargs -I _ cp gutenberg/_ guten
```

Printing the results.

```
echo "There are $(ls -l gutenberg | wc -l) books that represent: $(du -sh gutenberg)"
```

3.8.2 Minimum Setup

1. Install [Java 8] or greater.
    - Debain-based OS: `sudo apt install openjdk-8-jdk`
2. Install the [Scala Build Tool] (SBT)
    - Debian-based OS:

```
$ sudo apt install wget
$ wget https://dl.bintray.com/sbt/debian/sbt-1.2.6.deb
$ sudo dpkg -i sbt-1.2.6.deb
$ rm -r sbt-1.2.6.deb
$ sbt about
```

3.8.3 WordCount

Let's do a quick wordcount example on the dataset as a warm-up exercise:

```
import com.rhdzmota.presentations.Settings.S03
import com.rhdzmota.presentations.S03.config.Context
import org.apache.spark.sql._

object WordCount extends Context {
  import spark.implicits._

  final case class WordCount(word: String, count: Long)

  val data: Dataset[String] = spark.read.textFile(S03.Data.source)

  val wordcount: Dataset[WordCount] = data
    .flatMap(_.split("""\s+""")).map(_.toLowerCase.replaceAll("[^A-Za-z0-9]", "")).filte
    .groupByKey(identity).count().map({case (w, c) => WordCount(w, c)})
    .sort($"count".desc)

  def main(args: Array[String]): Unit = {
    println("S03 WordCount Application")
    wordcount.show()
    spark.close()
  }
}
```

Run:

```
WordCount.main(Array[String]())
```

Or:

```
sbt "content/runMain com.rhdzmota.presentations.S03.WordCount"
```

3.8.4 Next Word Prediction

The challenge we have consists con taking an n-set of books and create a model that's capable of predicting the next word given a context of the last m-words.

A similar approach is performed for generating word embeddings based on the distributional hypothesis - words that appear in the same contexts share semantic meaning.

3.8.5 Openscoring Container

We can easily leverage Openscoring with Docker.

Consider the following `Dockerfile`:

```
FROM maven:3.5-jdk-8-alpine

RUN apk update && apk upgrade && apk add --no-cache bash ca-certificates wget openssh

RUN wget https://github.com/openscoring/openscoring/releases/download/1.4.3/openscoring-

ADD application.conf application.conf

ENTRYPOINT java -Dconfig.file=application.conf -jar openscoring-server-executable-1.4.3.

EXPOSE 8080

CMD []
```

And the following `application.conf` file:

```
application {
  // List of JAX-RS Component class names that must be registered
  componentClasses = [
    "org.openscoring.service.filters.NetworkSecurityContextFilter",
    "org.openscoring.service.filters.ServiceIdentificationFilter"
  ]
}

networkSecurityContextFilter {
  // List of trusted IP addresses. An empty list defaults to all local network IP addres
  // A client that originates from a trusted IP address (as indicated by the value of th
  trustedAddresses = ["*"]
}
```

We can create our custom image with:

```
docker build -t next-word-demo/openscoring resources/provided/docker/
```

Now we can run a docker container with:

```
docker run -p 8080:8080 -d --name next-word-engine next-word-demo/openscoring
```

You can test this service is running by going to: `http://{ip-address}:8080/openscoring` where `ip-address` is your `docker-machine ip` or `localhost`. We can now upload the resulting dataset to the Openscoring API:

```
curl -X PUT --data-binary @resources/output/model/2019-01-25T01-07-14.836-89d19488-3d3d-
-H "Content-type: text/xml" \
http://192.168.99.100:8080/openscoring/model/next-word
```

We should see the model in `http://{ip-address}:8080/openscoring/model/next-word-demo`.

Enjoy your scoring!

```
curl -X POST --data-binary @resources/provided/requests/req-01.json \
    -H "Content-type: application/json" \
    http://192.168.99.100:8080/openscoring/model/next-word \
    | jq '.result."pmml(prediction)"'
```

3.9 References

References

Karau, Holden, and Rachel Warren. 2017. *High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark*. " O'Reilly Media, Inc."

Laskowski, Jacek. 2017. "Mastering Apache Spark." *Gitbook: Https://Jaceklaskowski.gitbooks.io/Mastering Apache-Spark* 25.

Lublinsky, Boris. 2017. *Serving Machine Learning Models - a Guide to Architecture, Stream, Processing Engines, and Frameworks*. "O'Reilly Media, Inc."

Mac Lane, Saunders. 2013. *Categories for the Working Mathematician*. Vol. 5. Springer Science & Business Media.

Meijer, Erik. 2015. "Category Theory, the Essence of Interface-Based Design." Youtube - FooCafe. https://www.youtube.com/watch?v=JMP6gI5mLHc.

Odersky, Martin. 2018. "New Functional Constructs in Scala 3." Youtube - FunctionalTV. https://www.youtube.com/watch?v=6P06YHc8faw.

Pierce, Benjamin C, and C Benjamin. 2002. *Types and Programming Languages*. MIT press.

Ruusmann, Villu. 2017. "R, Scikit-Learn, and Apache Spark Ml: What Difference Does It Make?" Youtube - StartApp. https://www.youtube.com/watch?v=CdVXEQgmnfY.

Tan, Lauren. 2018. "Learning to Love Type Systems." Youtube - DotJs. https://www.youtube.com/watch?v=cj07Fwzamy0.

Wampler, Dean. 2017. "Fast Data Architectures for Streaming Applications." Youtube - GOTO Conferences. https://www.youtube.com/watch?v=oCW5y4_8uGU.

Xie, Yihui, JJ Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. CRC Press.