Quantmetry

Data Science Consulting

Model as code

How to automate the deployment of R models

in production environment

June, 2nd

Matthieu Vautrot
mvautrot@quantmetry.com
Isabelle Robin
irobin@quantmetry.com

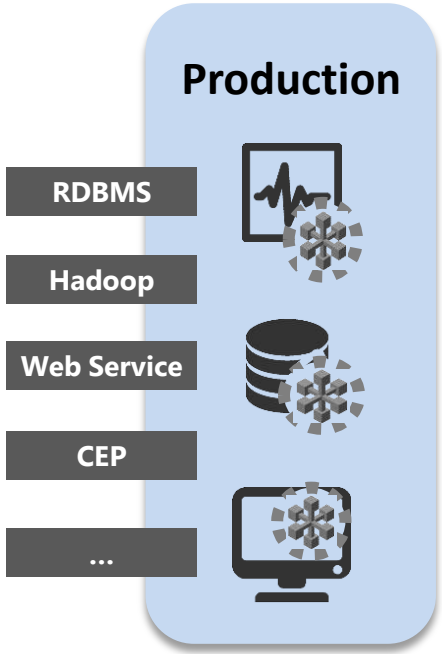
Isabelle Robin
Data scientist

@IsabelleRobin3

Matthieu Vautrot
Big Data & Analytics

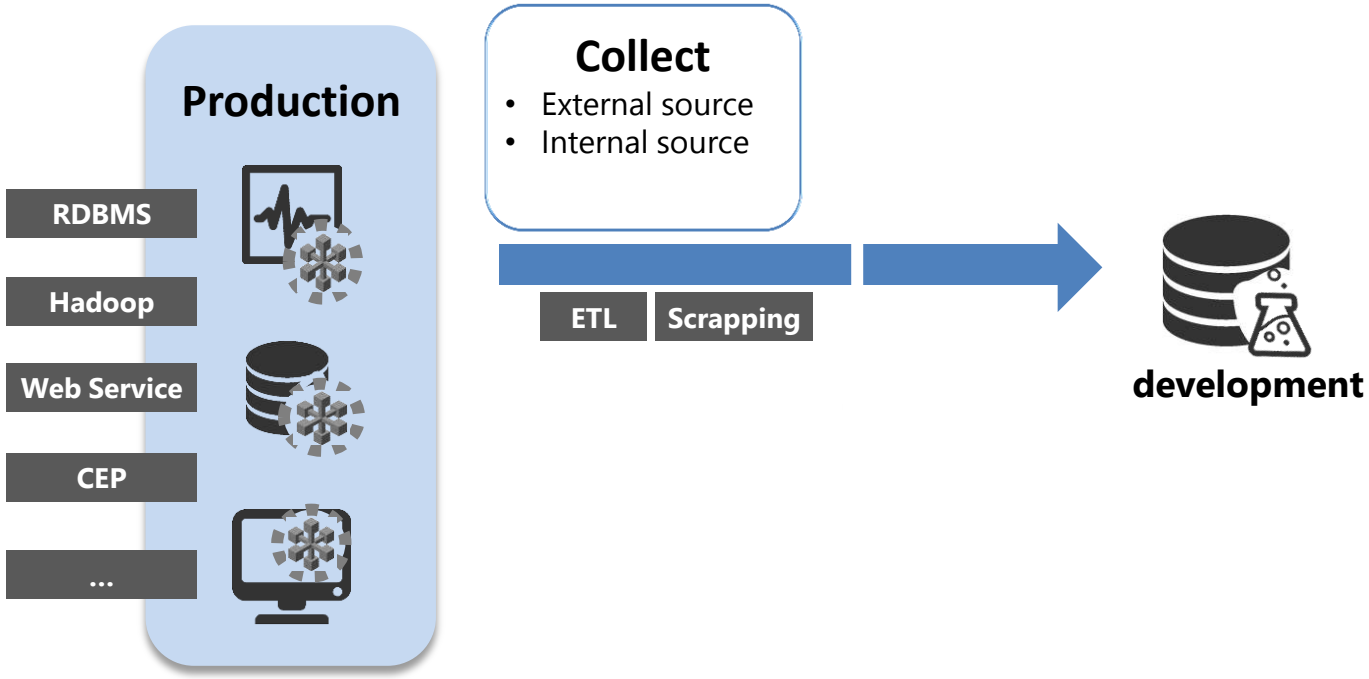
@matthieuvautrot

Analytical project organization

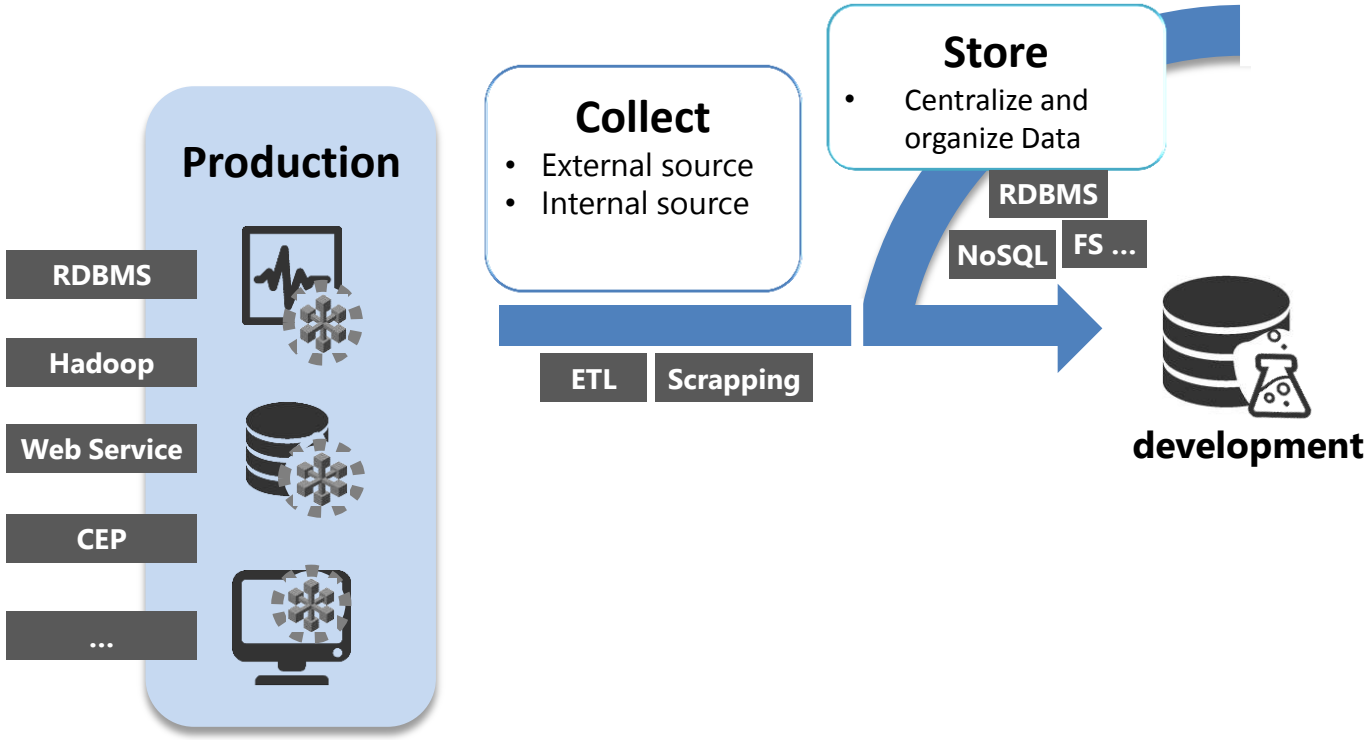


development

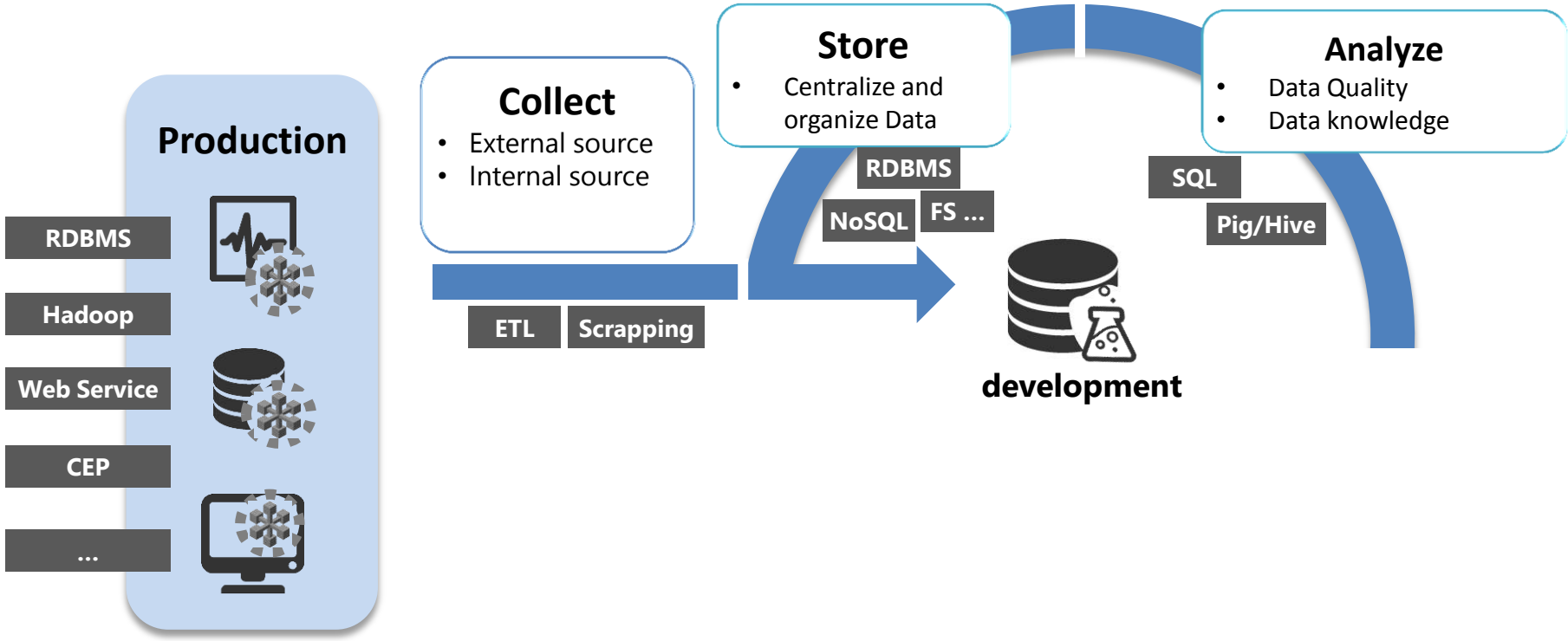
Analytical project organization



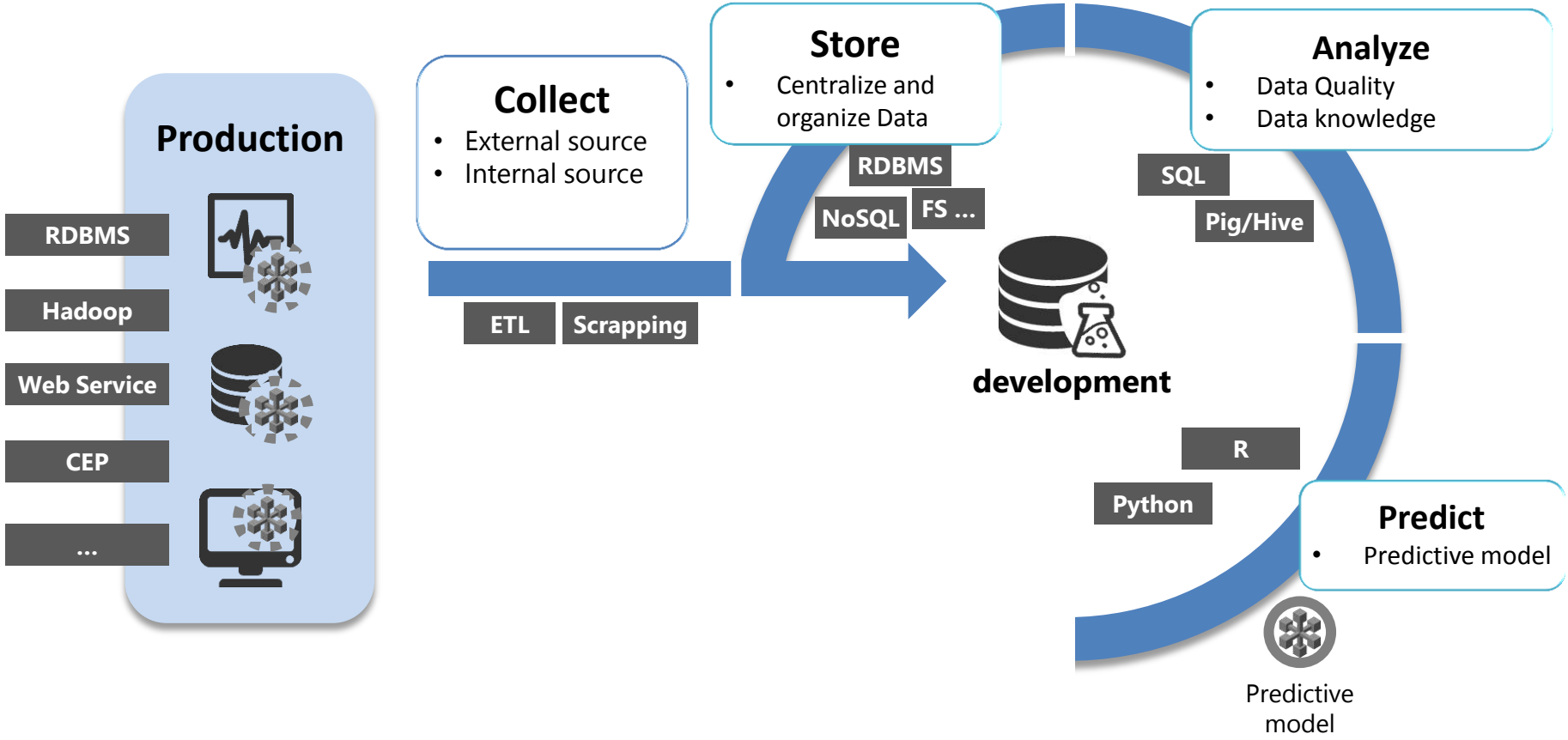
Analytical project organization



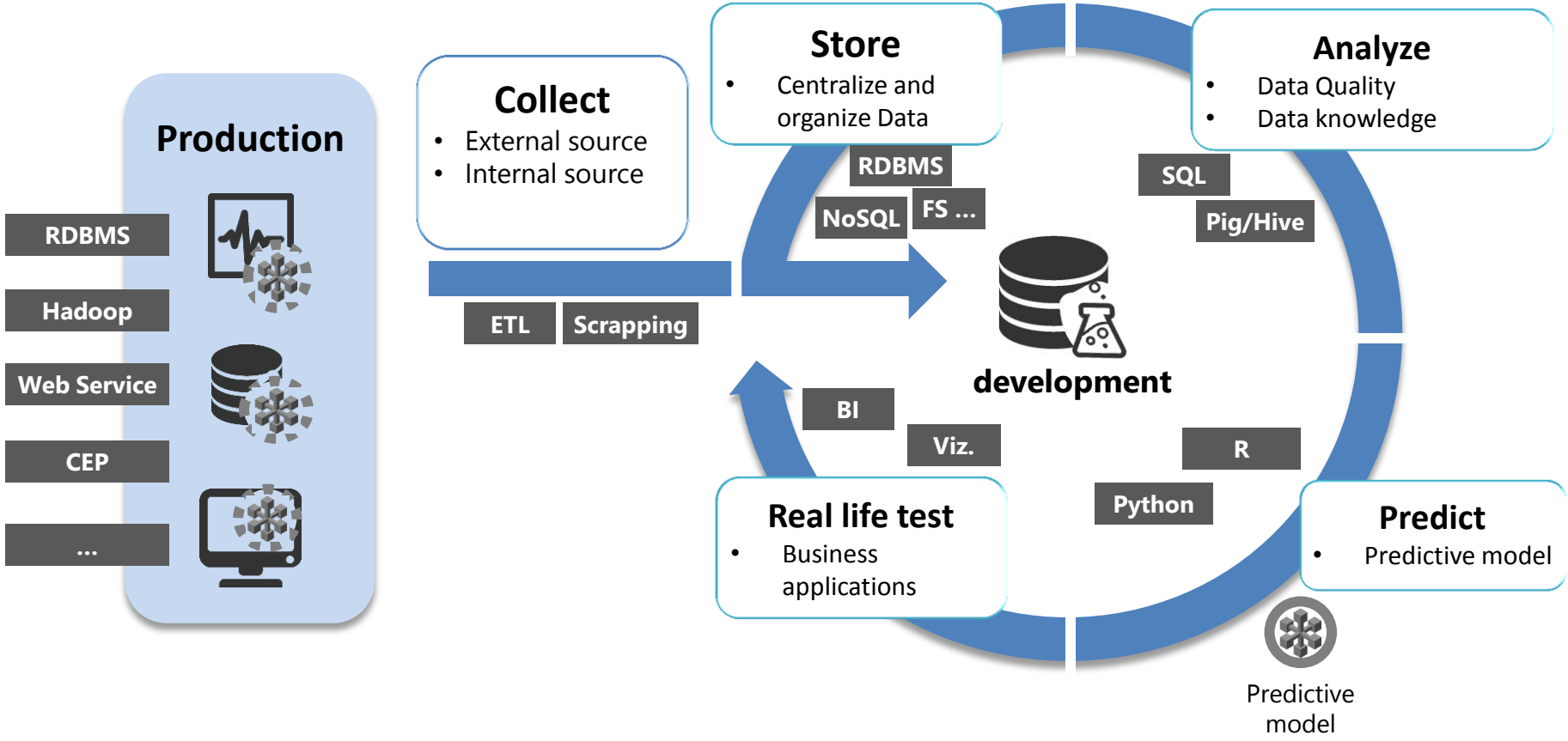
Analytical project organization



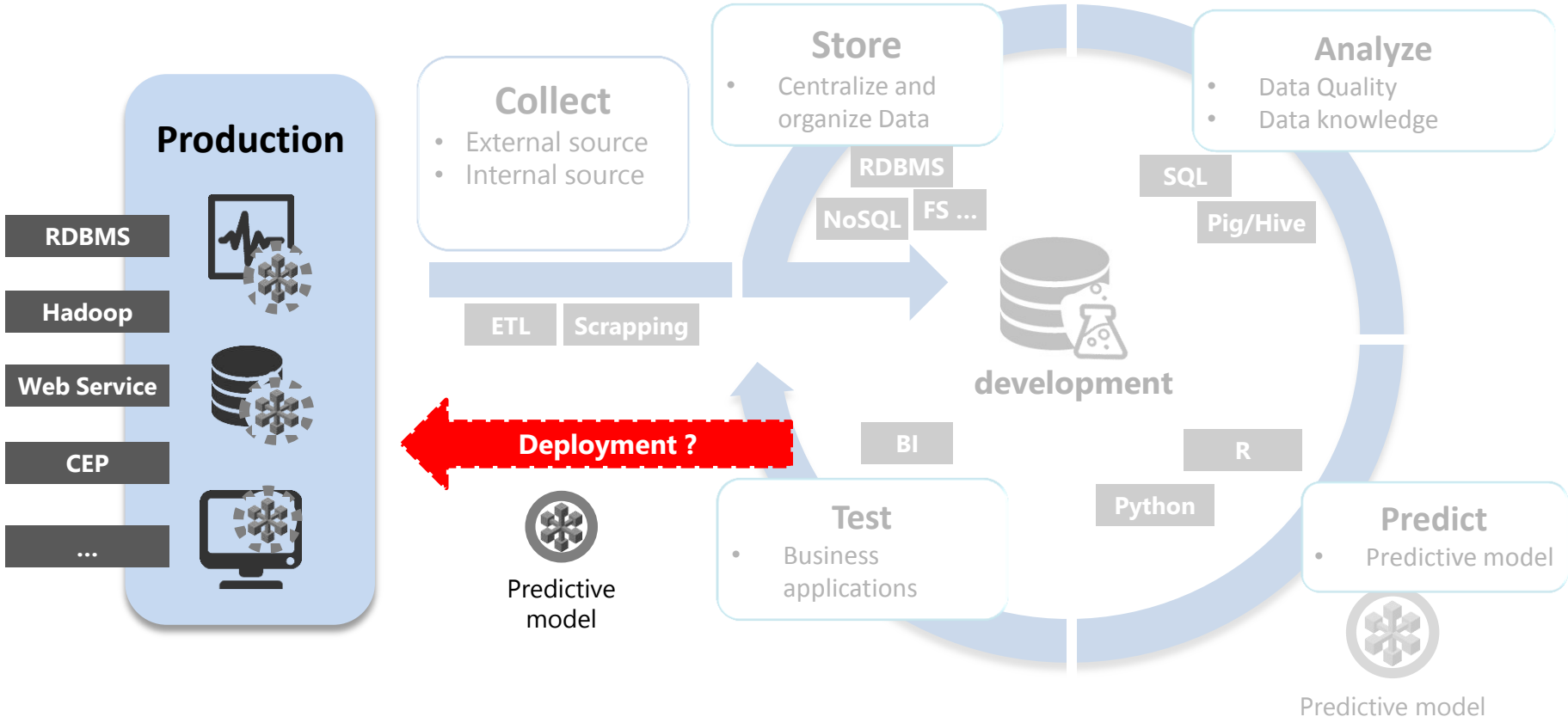
Analytical project organization



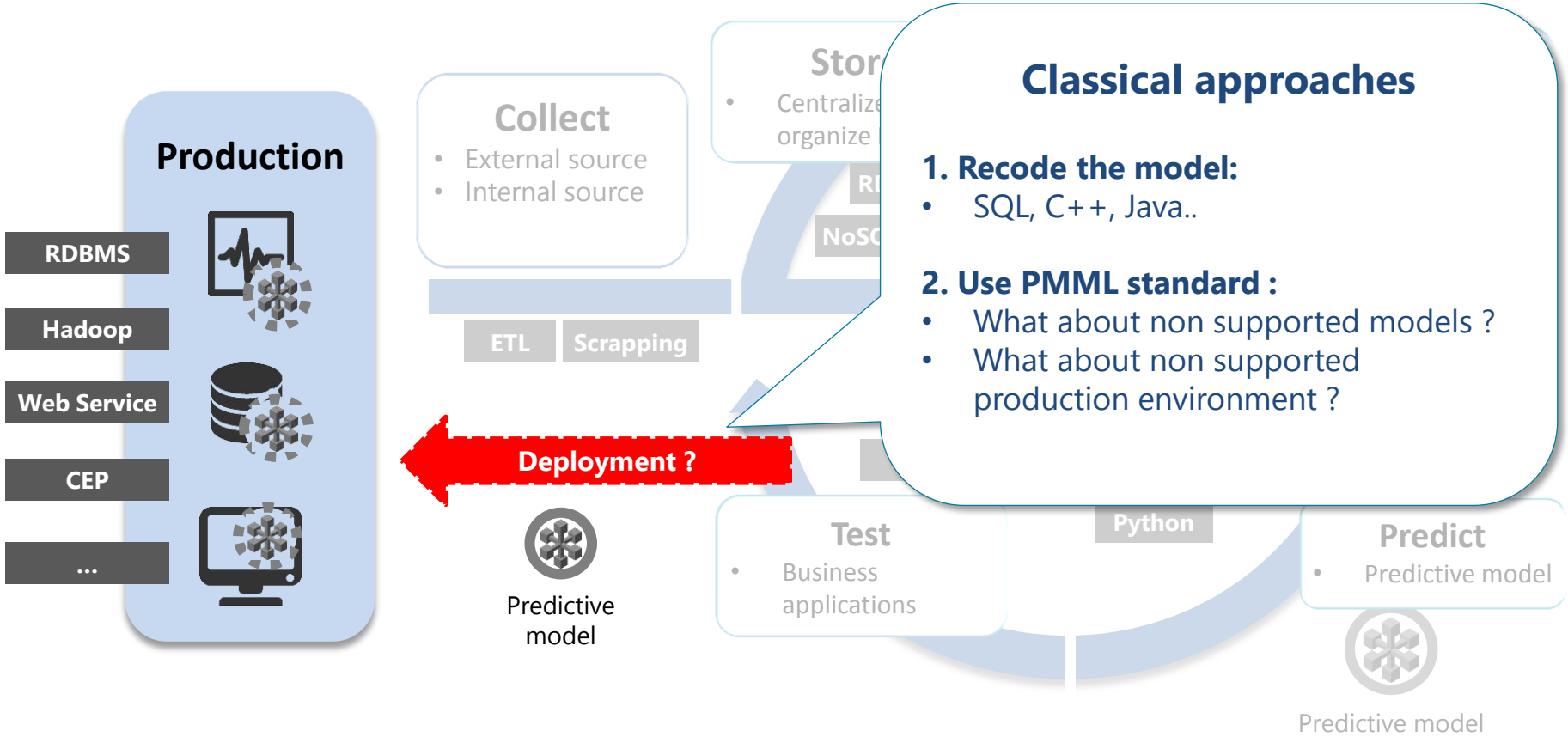
Analytical project organization



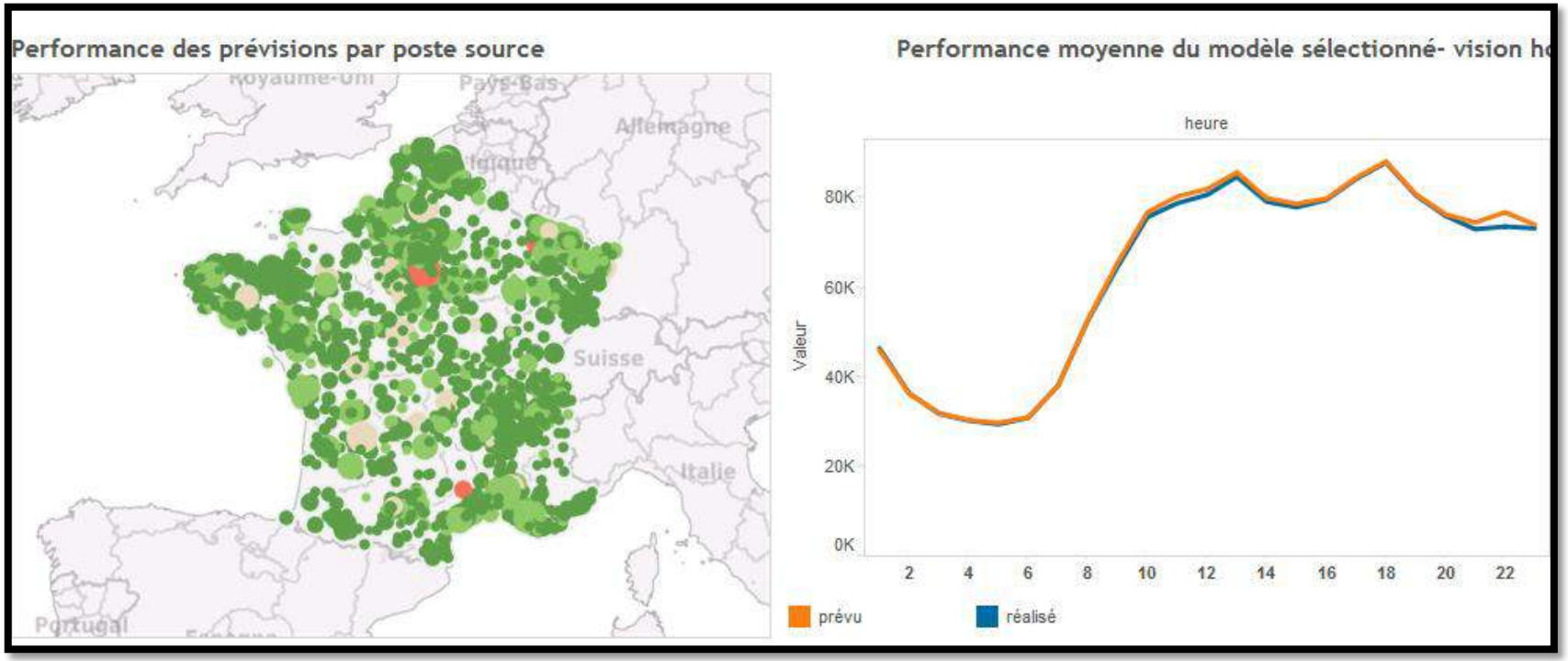
Analytical project organization



Analytical project organization

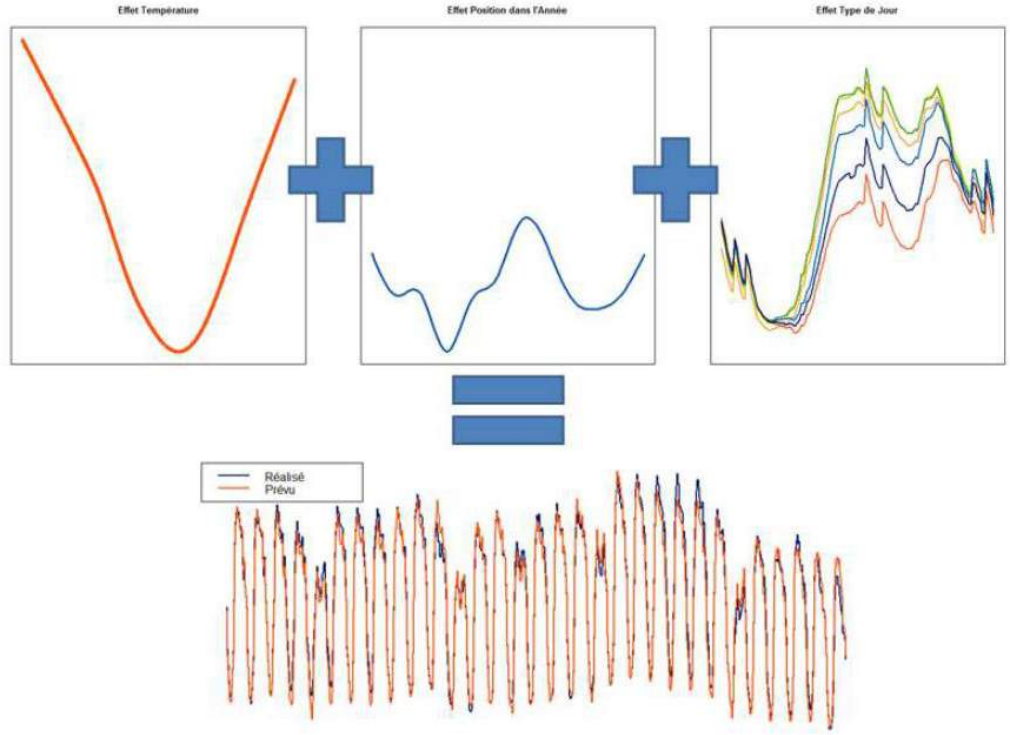


Who did it bother ?
Predictive model for charge consumtion



A bit of GAM : Generalized additive models

$$y_t = f_1(T_t) + f_2(I_t) + f_3(H_t) + \varepsilon_t$$



- From specific R package {mgcv}
- Not supported by PMML
- Results are not easily writable

An other approach

Model as code :

Directly deploy and use the R object for prediction

Three goals in mind :

1. **Drastically reduce models deployment time**
2. **General approach :** for one environment use the same code for any model
3. **Stable in performance**

R models outside of R

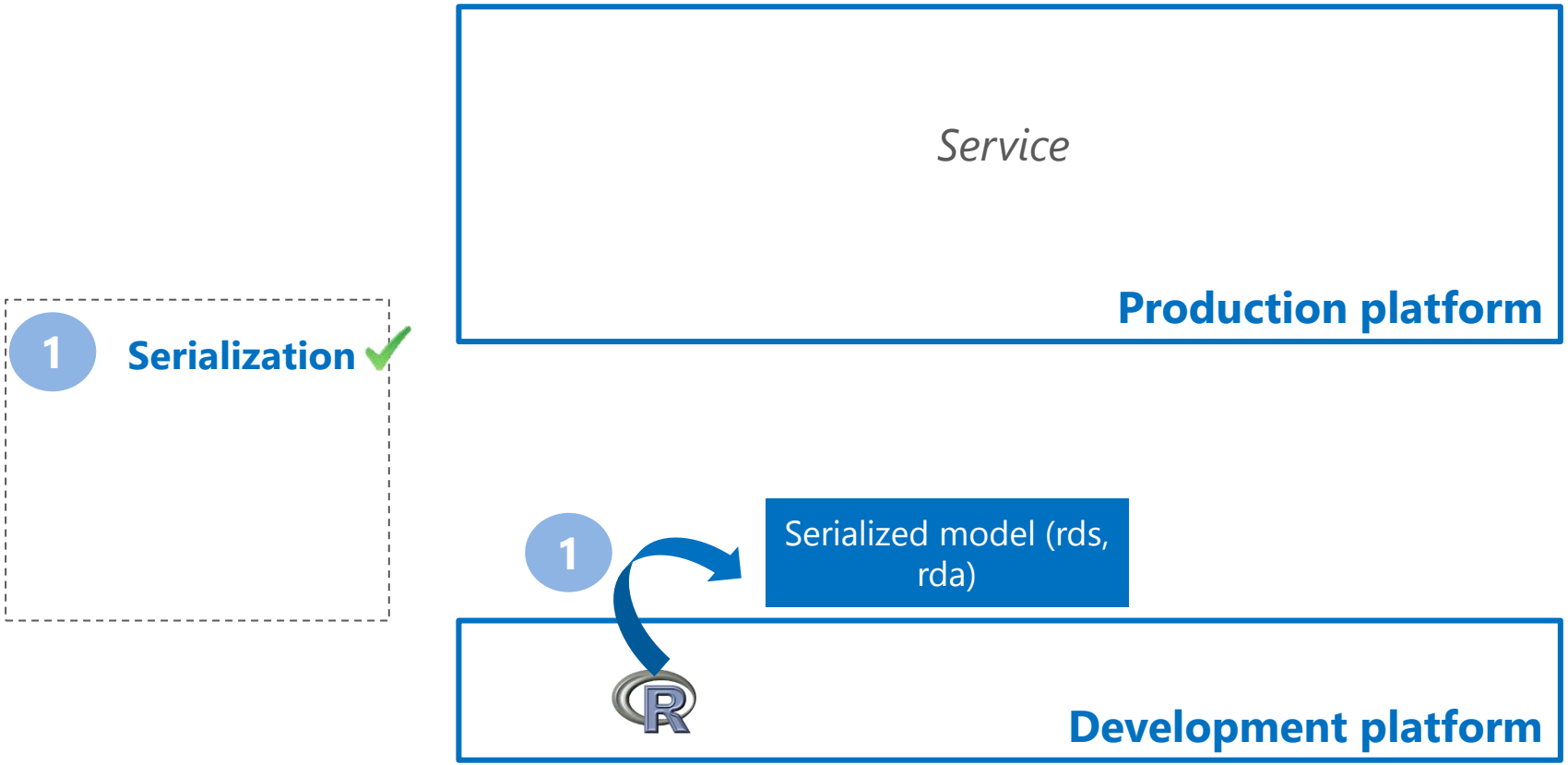
General description

Service

Production platform

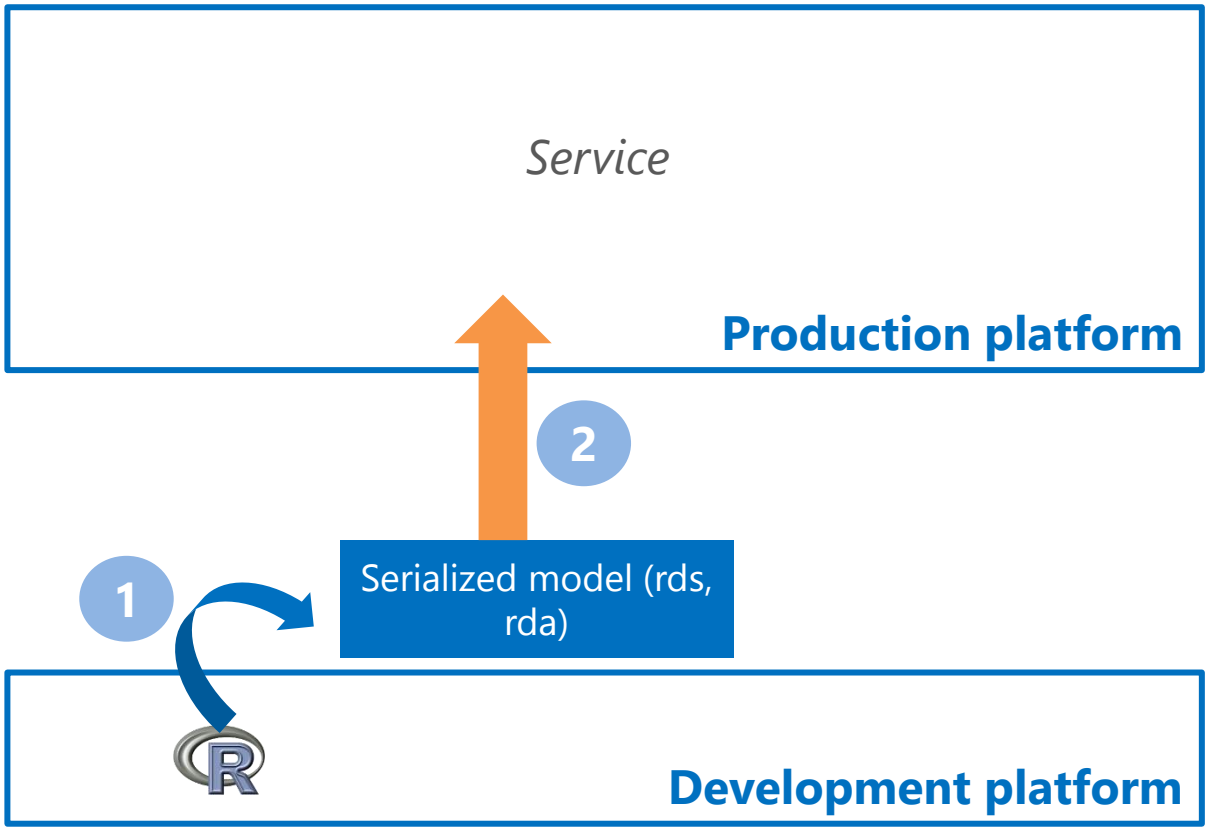
Development platform

General description



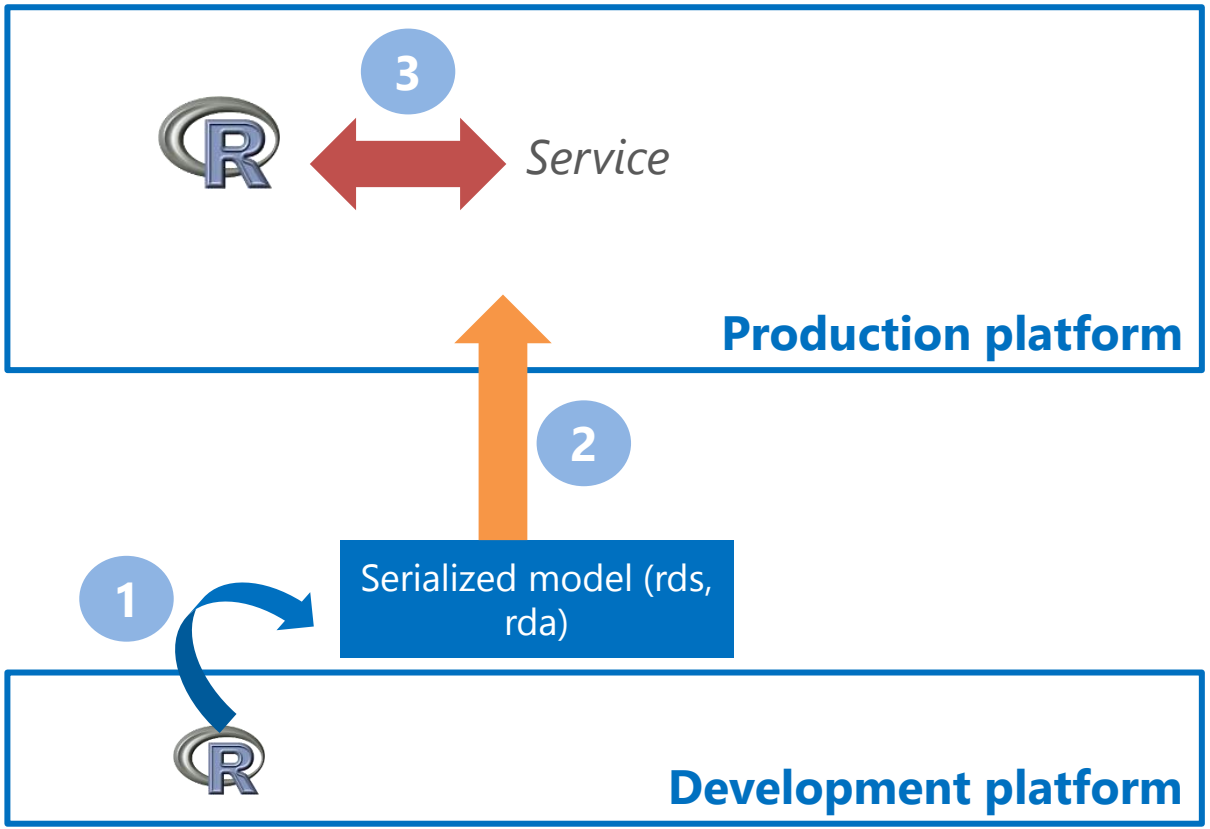
General description

1. Serialization ✓
2. Deployment ✓



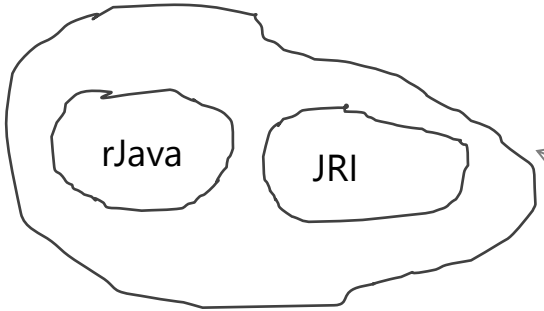
General description

1. Serialization ✓
2. Deployment ✓
3. Prediction ?



Communication between Java and R

- **rJava library** merges two projects :
  - JRI which enables to open a R session in Java
  - rJava which enables to use Java in a R session



And the whole thing is called rJava !

- **1 - Start R engine** : `Rengine engine = Rengine.getMainEngine();`
- **2 - Use any R command** : `rengine.eval("dt <- read.csv('myfile.csv')");`

Parameter is a string of *code*

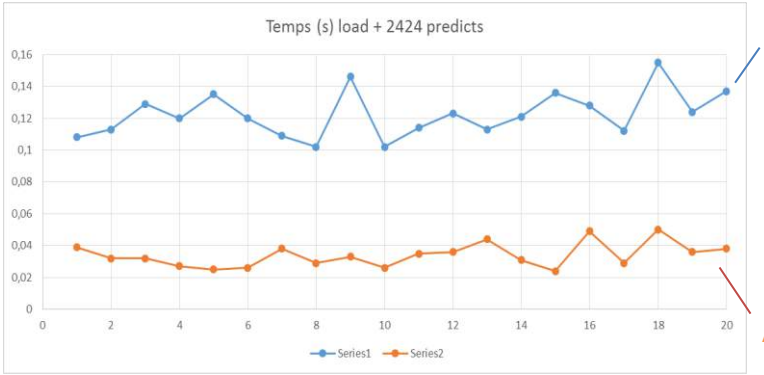
On our way to prediction

Some R models seem to carry this much

On our way to prediction

- Some R models become quite heavy because they **carry data used for training** !

- Lighten models by **removing all this meta-data** **=**
  - ➢ Slightly faster prediction
  - ➢ Gain huge storage and memory space

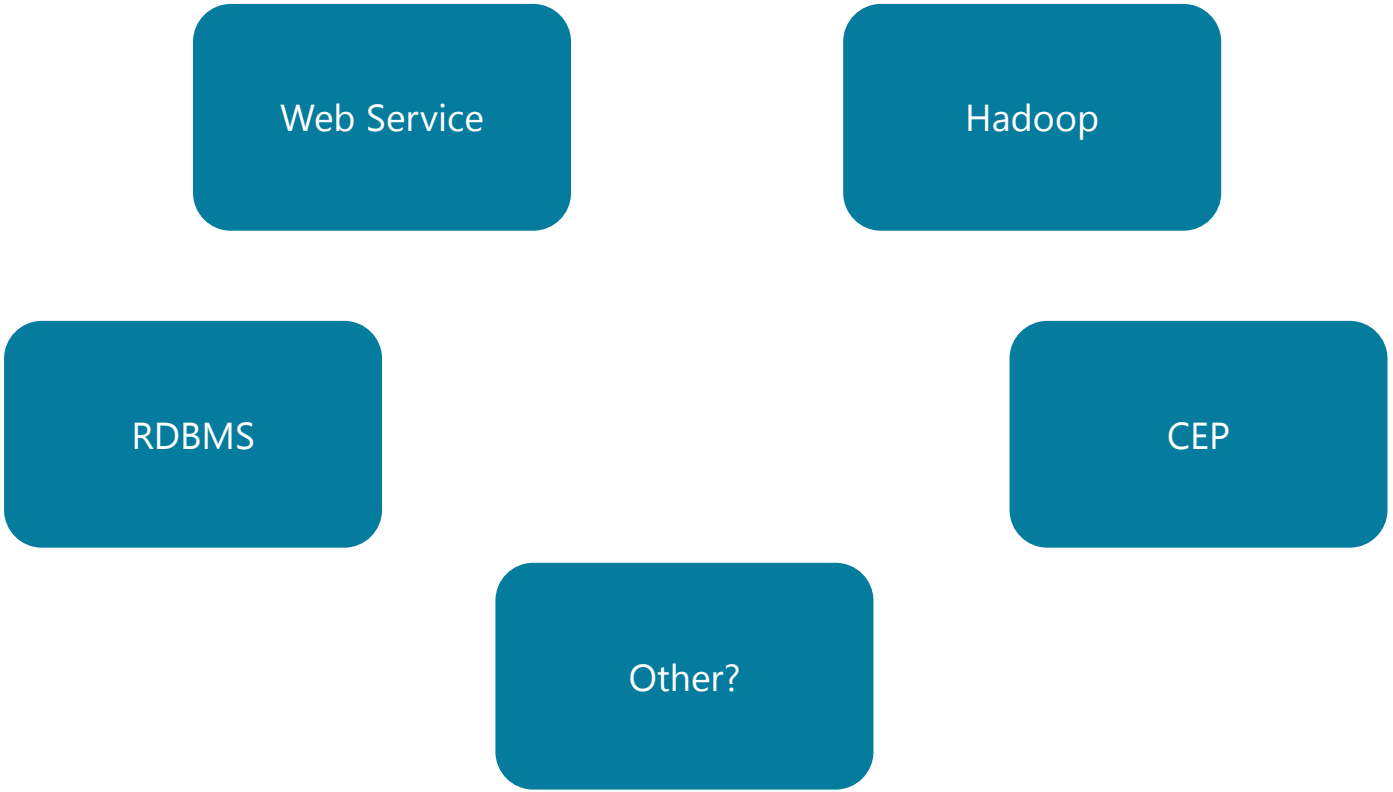


Before dehydration

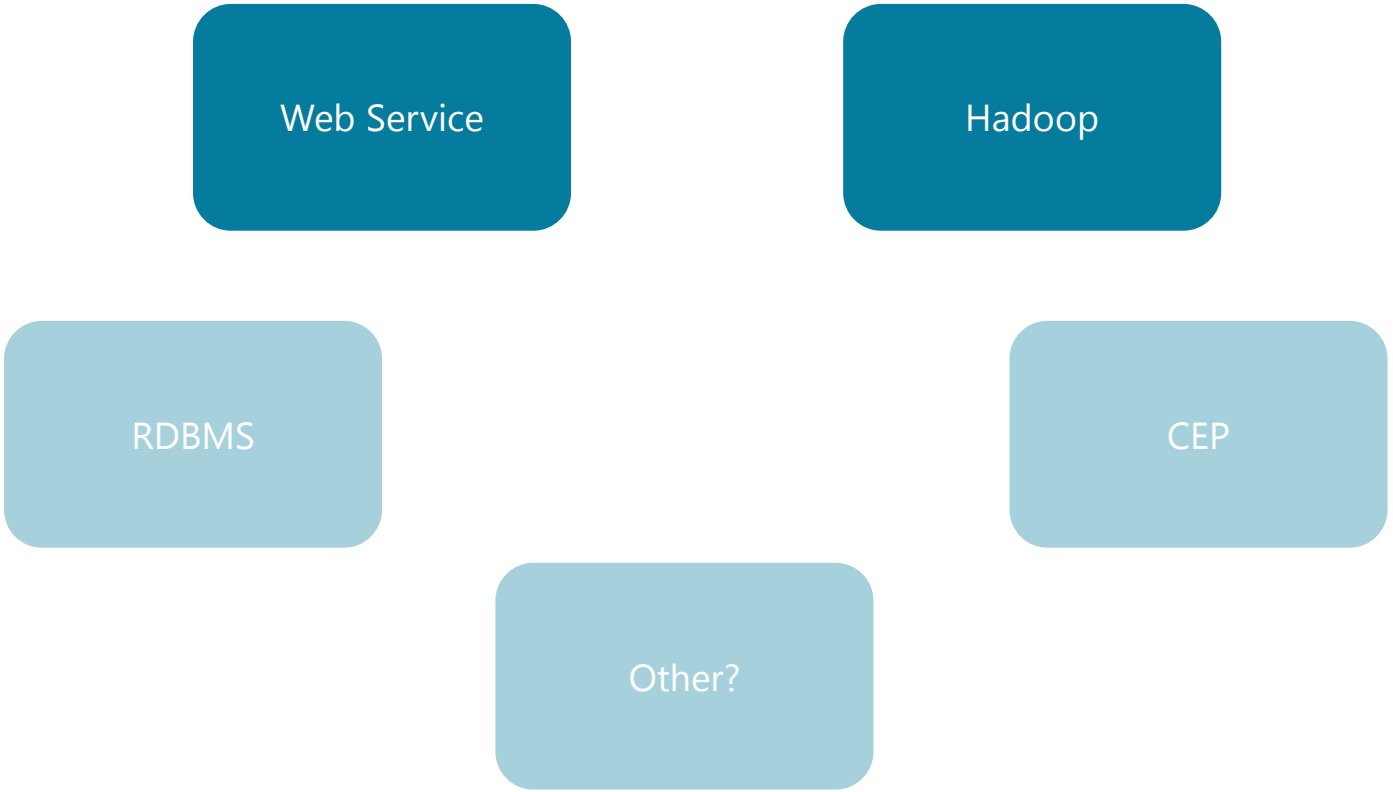
After dehydration

Crédit photo : Caroline Schroeder

Should work on multiple production environment



Should work on multiple production environment



A bit of REST

Workflow summary

1. 

   Build a predictive model

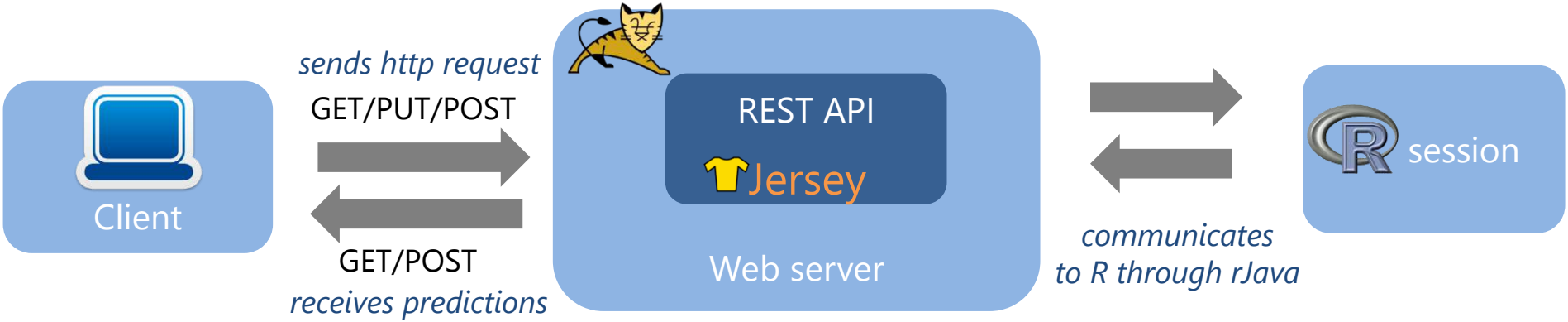
2. 

   Serialize and deploy it to the API (**PUT** request)

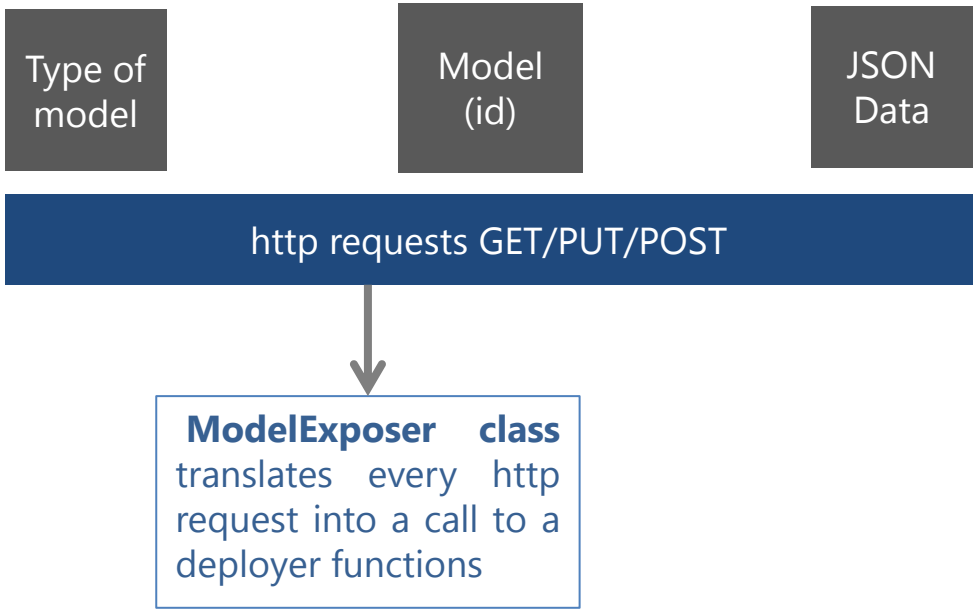
3. 

   Launch prediction on new data (**POST** request)

Architecture

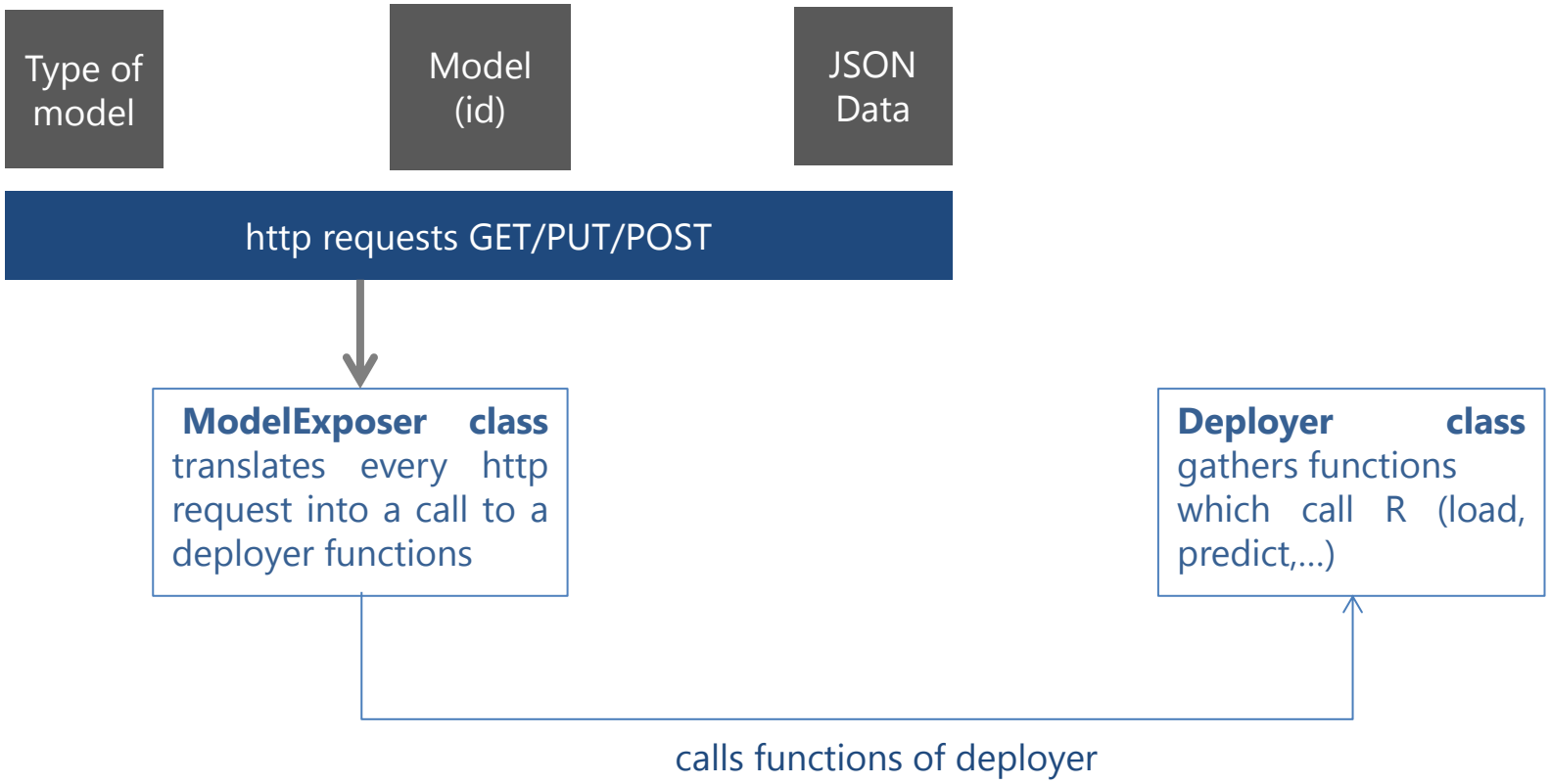


Class diagram



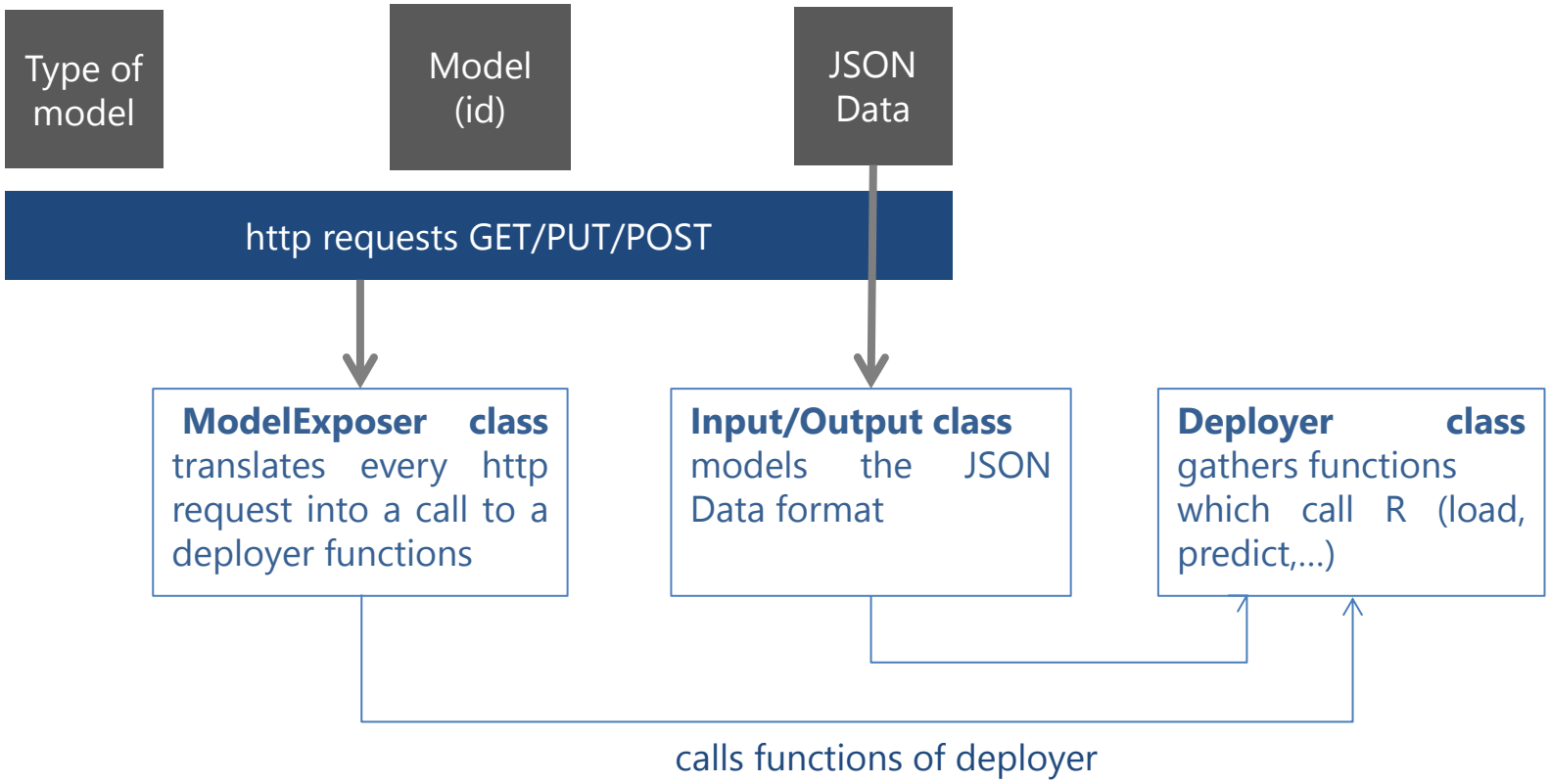
Very similar architecture to openscoring's : http://github.com/openscoring

Class diagram



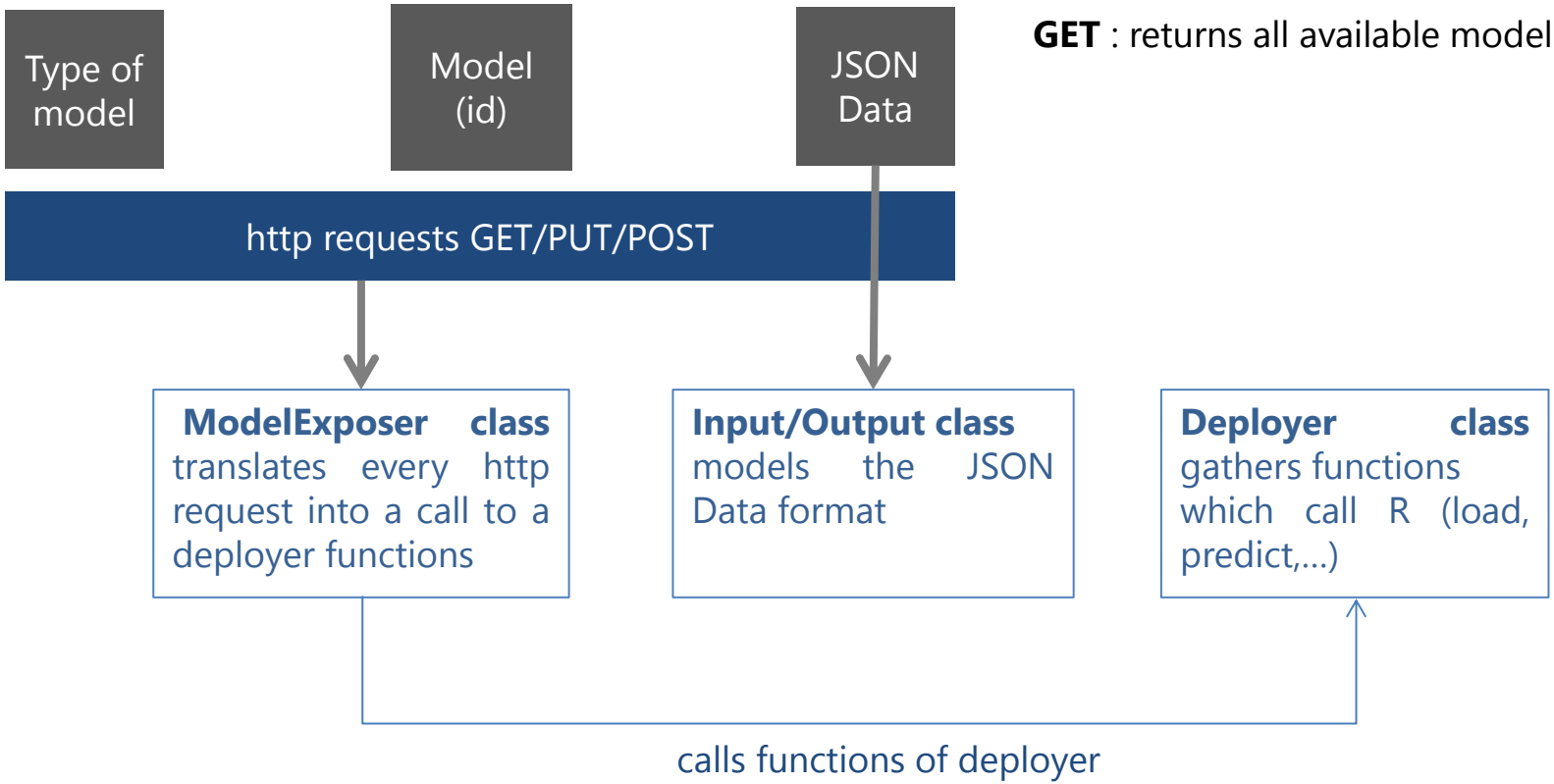
Very similar architecture to openscoring's : http://github.com/openscoring

Class diagram



Very similar architecture to openscoring's : http://github.com/openscoring

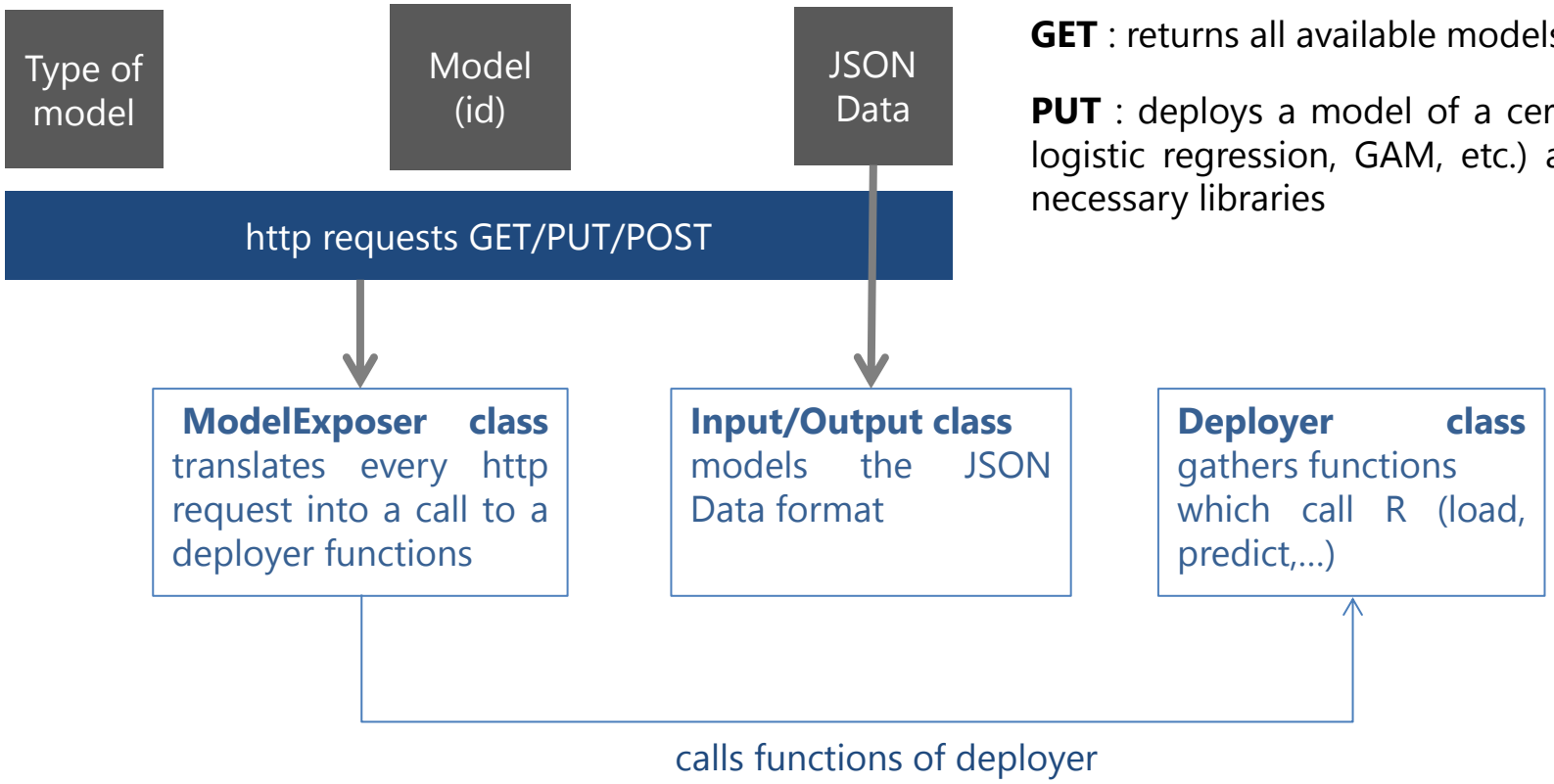
Class diagram



GET : returns all available models

Very similar architecture to openscoring's : http://github.com/openscoring

Class diagram

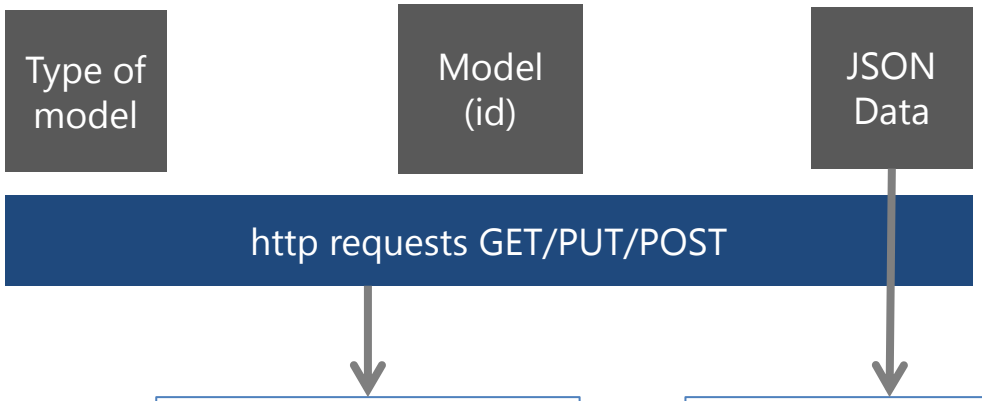


GET : returns all available models

PUT : deploys a model of a certain type (i.e logistic regression, GAM, etc.) and load the necessary libraries

Very similar architecture to openscoring's : http://github.com/openscoring

Class diagram



GET : returns all available models

PUT : deploys a model of a certain type (i.e logistic regression, GAM, etc.) and load the necessary libraries

POST : returns the prediction of a given model on given data

ModelExposer class translates every http request into a call to a deployer functions

Input/Output class models the JSON Data format

Deployer class gathers functions which call R (load, predict,...)

calls functions of deployer

Very similar architecture to openscoring's : http://github.com/openscoring

POST – Read the data given

It looks like that:

```
{  "id" : "myID",
    "arguments" :
                    { "X" : [x1, x2, x3],
                      "Y" : [y1, y2, y3]
                    }
}
```

Several features possible in the arguments nested JSON

Multiple rows are converted to array

x1, y1, … are double values

POST – Read the data given

It looks like that:

```
{  "id" : "myID",
    "arguments" :
                    { "X" : [x1, x2, x3],
                      "Y" : [y1, y2, y3]
                    }
}
```

Several features possible in the arguments nested JSON

Multiple rows are converted to array

x1, y1, … are double values

In R, load with RJSONIO library and then it's converted to R dataframe :

```
json_file <- fromJSON(jsonData)
json_file <- lapply(json_file, function(x) {
  x[sapply(x, is.null)] <- NA
  unlist(x)
})
newdata <- as.data.frame(t(do.call("rbind", json_file)))
```

POST – Towards prediction

Load the model

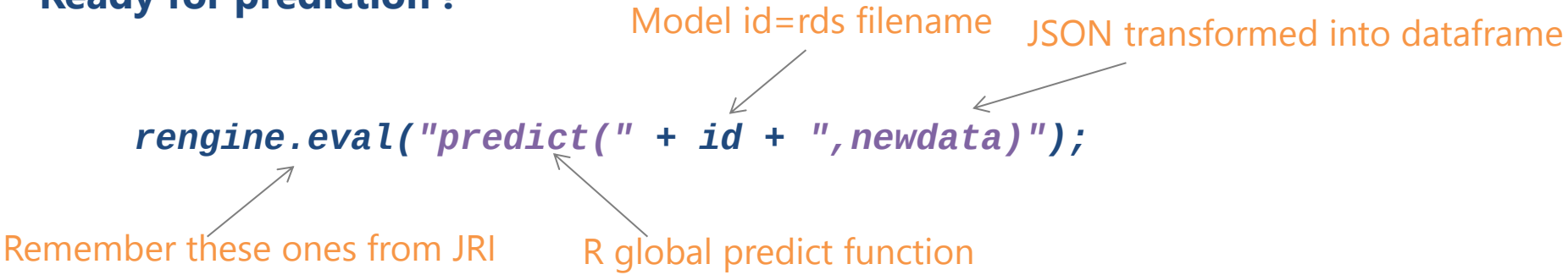
- Load specific libraries according to type
- Then use R function readRDS to load the model

POST – Towards prediction

Load the model

- Load specific libraries according to type
- Then use R function readRDS to load the model

Ready for prediction !



POST – Towards prediction

Load the model

- Load specific libraries according to type
- Then use R function readRDS to load the model

Ready for prediction !

Model id=rds filename

JSON transformed into dataframe

```
rengine.eval("predict(" + id + ",newdata)");
```

Remember these ones from JRI

R global predict function

Convert your dataframe result into a JSON output

POST – Towards prediction

Load the model

- Load specific libraries according to type
- Then use R function readRDS to load the model

Ready for prediction !

Model id=rds filename

JSON transformed into dataframe

```
rengine.eval("predict(" + id + ",newdata)");
```

Remember these ones from JRI

R global predict function

Convert your dataframe result into a JSON output

A Big Data approach

Streaming or encapsulation ?

Hadoop (Pig, Hive) streaming :

- Not a lot to set up
- Easy to implement
- Use standard I/O

```
hadoop jar $HADOOP_PATH/hadoop-
streaming-XXX.jar \
-input dir_input \
-output dir_output  \
-mapper script.r \
-files script.r, model.rds
```

Streaming or encapsulation ?

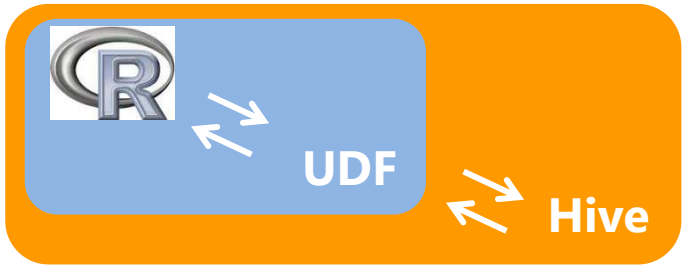
Hadoop (Pig, Hive) streaming :

- Not a lot to set up
- Easy to implement
- Use standard I/O

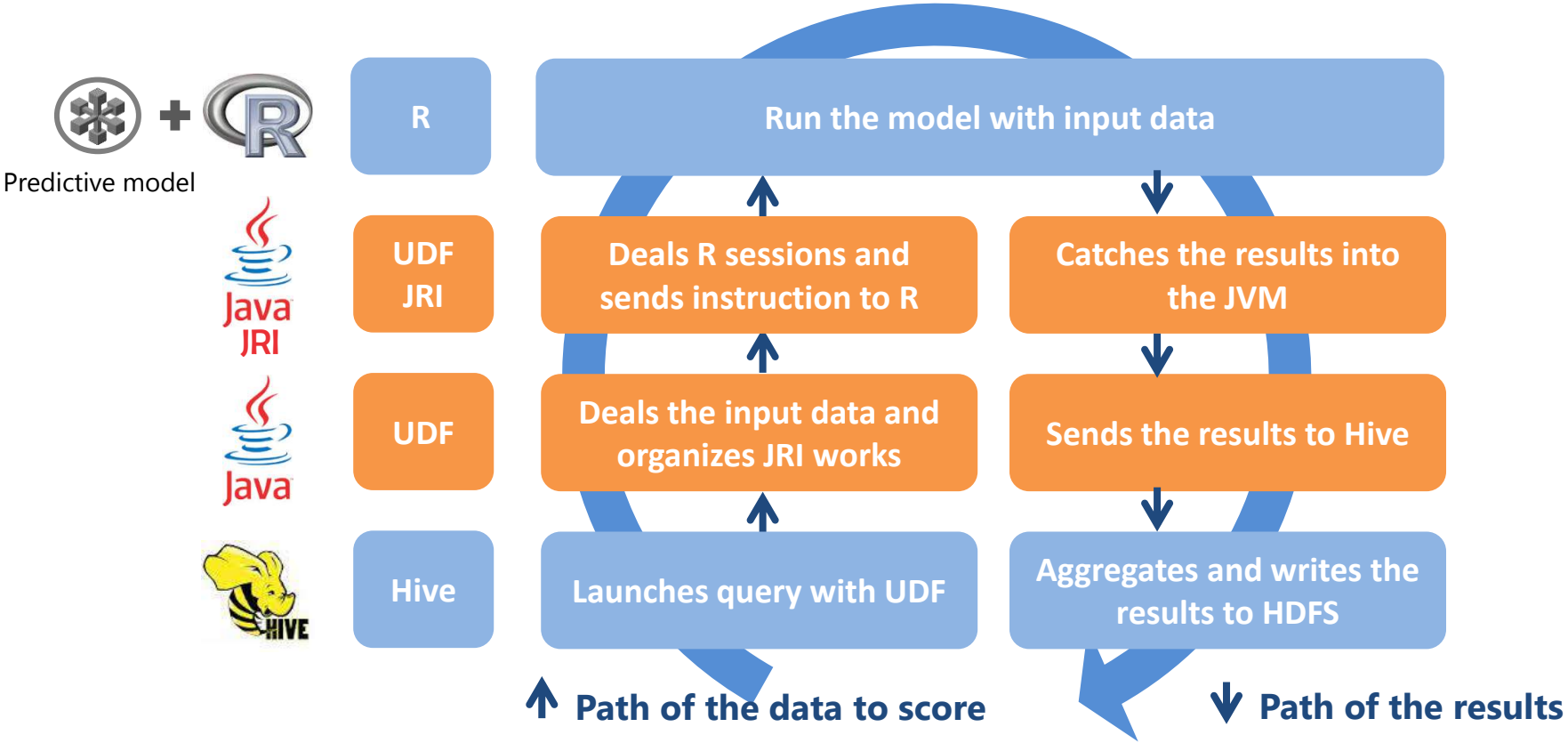
```
hadoop jar $HADOOP_PATH/hadoop-
streaming-XXX.jar \
-input dir_input \
-output dir_output  \
-mapper script.r \
-files script.r, model.rds
```

Encapsulation (MR, or UDF):

- Better control on data input
- Java code that runs java code
- Uses JRI to communicate between R and Java



UDF data cycle



Hive UDF or Hive generic UDF ?

Hive UDF : …

Hive generic UDF - a bit more complex to write but …

- Deal dynamic number of parameters
- Deal better NULL value
- Deal constant parameter better

Hive UDF or Hive generic UDF ?

Hive UDF : …

Hive generic UDF - a bit more complex to write but …

- Deal dynamic number of parameters
- Deal better NULL value
- Deal constant parameter better

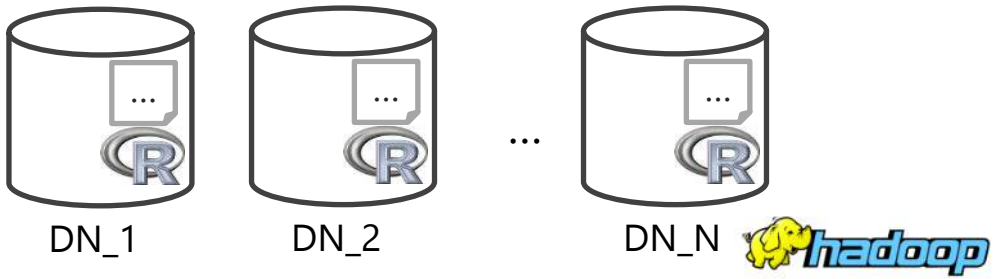
Best practice :

```
import org.apache.hadoop.hive.ql.udf.generic.GenericUDF;
```

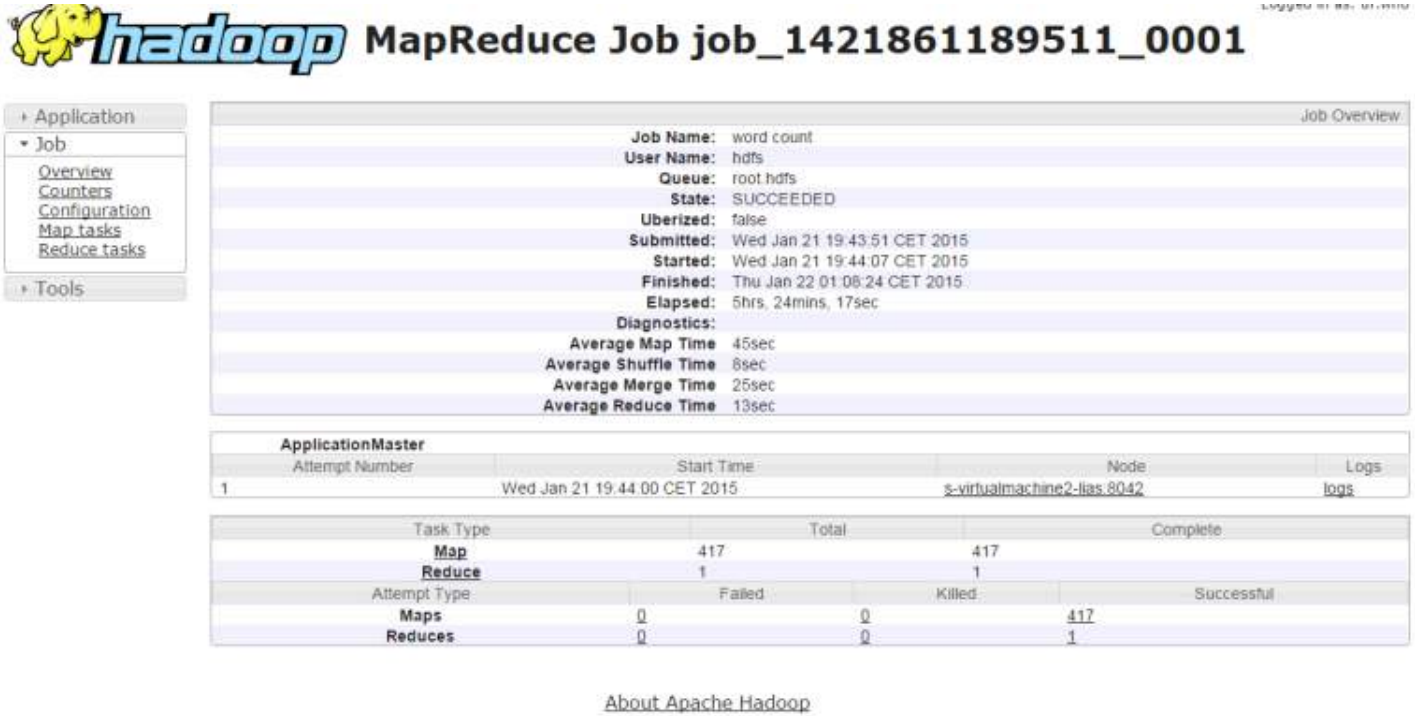
First thing first : set up the nodes

Once for each node :

1. Install **R** and **rJava** (+JRI) ;
2. Install the models required libraries {mgcv} ;
3. Set environment variable (**$R_HOME**, **jri** to **$LD_LIBRARY_PATH**) ;
4. Symbolic link some **.so** file from jri to hadoop native lib ;



Use logs !


MapReduce Job job_1421861189511_0001
Application
Job
Overview
Counters
Configuration
Map tasks
Reduce tasks
Tools
Job Overview
Job Name: word count
User Name: hdfs
Queue: root.hdfs
State: SUCCEEDED
Uberized: false
Submitted: Wed Jan 21 19:43:51 CET 2015
Started: Wed Jan 21 19:44:07 CET 2015
Finished: Thu Jan 22 01:08:24 CET 2015
Elapsed: 5hrs, 24mins, 17sec
Diagnostics:
Average Map Time 45sec
Average Shuffle Time 8sec
Average Merge Time 25sec
Average Reduce Time 13sec
ApplicationMaster
Attempt Number
Start Time
Node
Logs
1
Wed Jan 21 19:44:00 CET 2015
s-virtualmachine2-lias:8042
logs
Task Type
Total
Complete
Map
417
417
Reduce
1
1
Attempt Type
Failed
Killed
Successful
Maps
0
0
417
Reduces
0
0
1
About Apache Hadoop

Workflow summary

1

Build a predictive model

2

Serialize and deploy the model

3

Launch prediction on new data

UDF – maps parameters and values

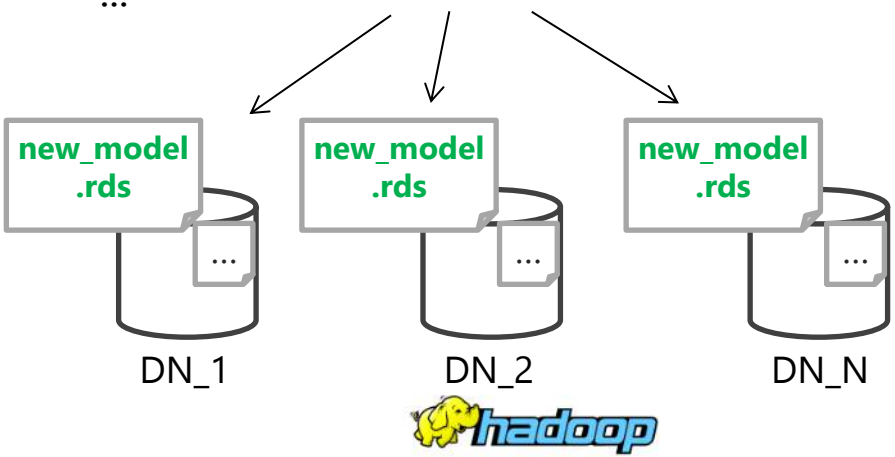
```
new_model <- model(formula='y~R1+R2')
```

File spreading – distributed cash

```
new_model <- model(formula='y~R1+R2')
```

HQL

```
add file new_model.rds;
…
```



Workflow summary

1. 
   Build a predictive model

2. 
   Serialize and deploy the model

3. 
   Launch prediction on new data

UDF – maps parameters and values

```
new_model <- model(formula='y~R1+R2')
```

```
data <-dataframe(R1=f1,R2=f2)

…

predict(new_model, data);
```

UDF – maps parameters and values

```
new_model <- model(formula='y~R1+R2')
```

```
SELECT my_udf(
 "new_model:R1+R2",
 f1, f2
)
FROM data_table ;
```

UDF

```
data <-dataframe(R1=f1,R2=f2)

…

predict(new_model, data);
```

The Hive query

```
add jar JRI.jar genUdf.jar;
add file new_model.rds;

SELECT my_udf(
 "new_model:R1+R2",
 f1, f2
)
FROM table_data ;
```

Demo

Conclusion & perspectives

Two complementary approaches

Webservice : REST API

Number of rows	Data size (o)	Time (s)	Memory
1	33	-	OK
...	...	...	...
100,000	4,5M	35	OK
10 * 100,000	10*4.5M	381	OK

Hive UDF

Number of rows	Data size (o)	Nb. Mappers	Time (s)	Memory
1	34	1	51	OK
...	...	...	...	OK
10,000,000	441M	2	110	OK
100,000,000	4.4G	18	1312	OK

Two complementary approaches

Webservice : REST API

Number of rows	Data size (o)	Time (s)	Memory
1	33	-	OK
...	...	...	...
100,000	4,5M	35	OK
10 * 100,000	10*4.5M	381	OK

Hive UDF

Number of rows	Data size (o)	Nb. Mappers	Time (s)	Memory
1	34	1	51	OK
...	...	...	...	OK
10,000,000	441M	2	110	OK
100,000,000	4.4G	18	1312	OK

Cassandra ?

Storm ?

Spark ?

...

A word of python ?

A word of python ?

.rds	-> .pkl
JRI	-> r2py
Webservice	-> Django
…	

CRTL-F


Questions ?

Contacts

Jeremy Harroch
CEO
jharroch@quantmetry.com

Isabelle Robin
Data Scientist
irobin@quantmetry.com

Matthieu Vautrot
Big data & Analytics
mvautrot@quantmetry.com