



# APACHE SPARK

## ЕГО КОНФИГУРИРОВАНИЕ

АРТЕМ ПИЧУГИН, HEAD OF DATA-RELATED PROGRAMS



# Почему я здесь?

Artem Pichugin posted this



## How to configure Spark on a cluster with YARN

Artem Pichugin on LinkedIn  
January 20, 2016

[Edit](#) [Delete](#)

8,971 views of your article

Google  Search

Все Видео Картинки Покупки Новости Ещё Настройки Инструменты

Результатов: примерно 1 460 000 (0,40 сек.)

**Running Spark on YARN - Spark 2.4.0 Documentation - Apache Spark**  
<https://spark.apache.org/docs/latest/running-on-yarn.html> ▾ Перевести эту страницу  
Перейти к разделу **YARN-specific Kerberos Configuration** - To start the Spark Shuffle Service on each NodeManager in your YARN cluster, follow these instructions: Build Spark with the YARN profile. Locate the spark-<version>-yarn-shuffle.jar . Add this jar to the classpath of all NodeManager s in your cluster.

**Install, Configure, and Run Spark on Top of a Hadoop YARN Cluster**  
<https://www.linode.com/.../install-configure-run-spark-on-to...> ▾ Перевести эту страницу  
23 окт. 2017 г. - Configure Memory Allocation. Give Your YARN Containers Maximum Allowed Memory. Configure the Spark Driver Memory Allocation in Cluster Mode. Configure the Spark Application Master Memory Allocation in Client Mode. Configure Spark Executors' Memory Allocation. Integrate Spark with YARN · Understand Client and ... · Configure Memory Allocation

**Spark on YARN · Mastering Apache Spark - Jacek Laskowski**  
<https://jaceklaskowski.gitbooks.io/mastering...spark/yarn/> ▾ Перевести эту страницу  
You can start spark-submit with --verbose command-line option to have some settings displayed, including YARN-specific. See spark-submit and YARN options.

**Configuring Spark on YARN - MapR**  
<https://mapr.com/docs/60/Spark/ConfigureSparkOnYarn.html> ▾ Перевести эту страницу  
Starting in MEP 4.0, after following the steps outlined in the sub-topics in this section, you must run configure.sh -R as the final step in the configuration process.

**How to configure Spark on a cluster with YARN - LinkedIn**  
<https://www.linkedin.com/.../how-configure-spark-cluster-ya...> ▾ Перевести эту страницу  
20 янв. 2016 г. - But this material will help you to save several days of your life if you are a newbie and you need to configure Spark on a cluster with YARN.

Конфигурирование Spark на YARN  
[Редактировать](#)

12:40 28.04.2017 @a-pichugin

+7 4511 7 25

FB: 0 VK: 0

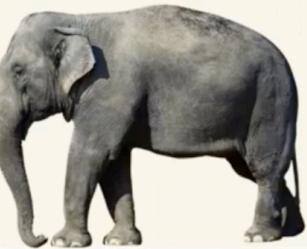
# Почему я здесь?



Артём Пичугин  
New Professions Lab



Предыстория



vilseev 12:59 PM ☆

Насколько я понял, по обсуждениям выше - слона тут никто не любит,



mephistopheles 4:23 PM

ВОТ 180к меня прям выводят из себя

Expand ▾

## Moscow Spark #2

<https://t.me/moscowspark>

# Предыстория

Что-то кластер не работает...

Слушатель

Мы

Аутсорсеры

Что-то кластер не работает...

## Предыстория

Экспертиза по Big Data должна быть in-house

# Предыстория

**Проблема:** 3-й запуск и 30 человек, недовольных тем, как работает Spark на кластере

## Предыстория

**Решение 1:** может быть, они что-то делают не так?

# Предыстория

Решение 1: может быть, они что-то делают не так?

Локально-то  
работает!

# Предыстория

**Решение 2:** найти готовый конфиг

## Предыстория

**Решение 2:** найти готовый конфиг

**Копипаст – зло!**

## Предыстория

**Решение 3:** читать документацию

## Предыстория

**Решение 3:** читать документацию

Не помогает!

# Предыстория

**Решение 4: неистово гуглить**

# Предыстория

Решение 4: неистово гуглить

Готовых ответов  
нет!

# Предыстория

**Решение 5:** уйти в это по уши

Помогает!

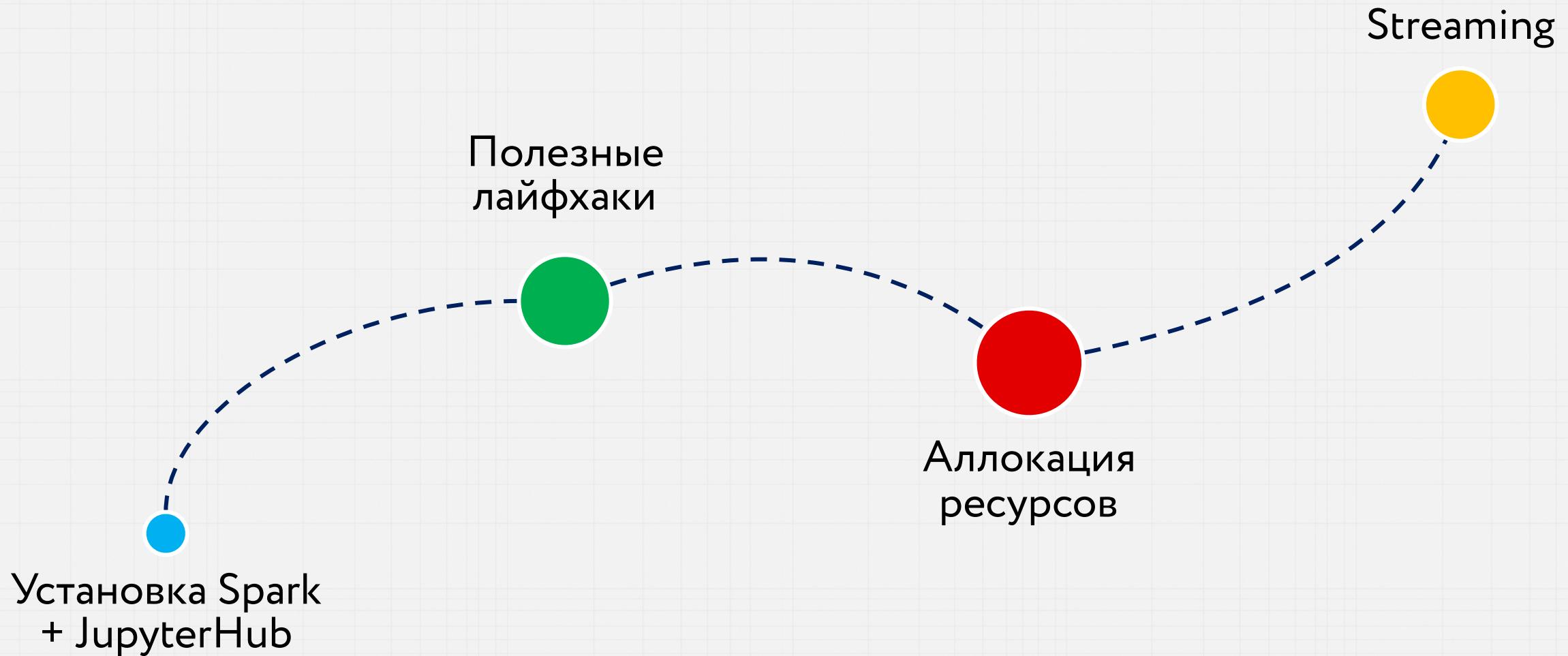
# Глобальная цель



Разверните Spark так, чтобы он был доступен для всех пользователей в нашей компании. Интегрируйте его с имеющимся у нас Hadoop.



# План





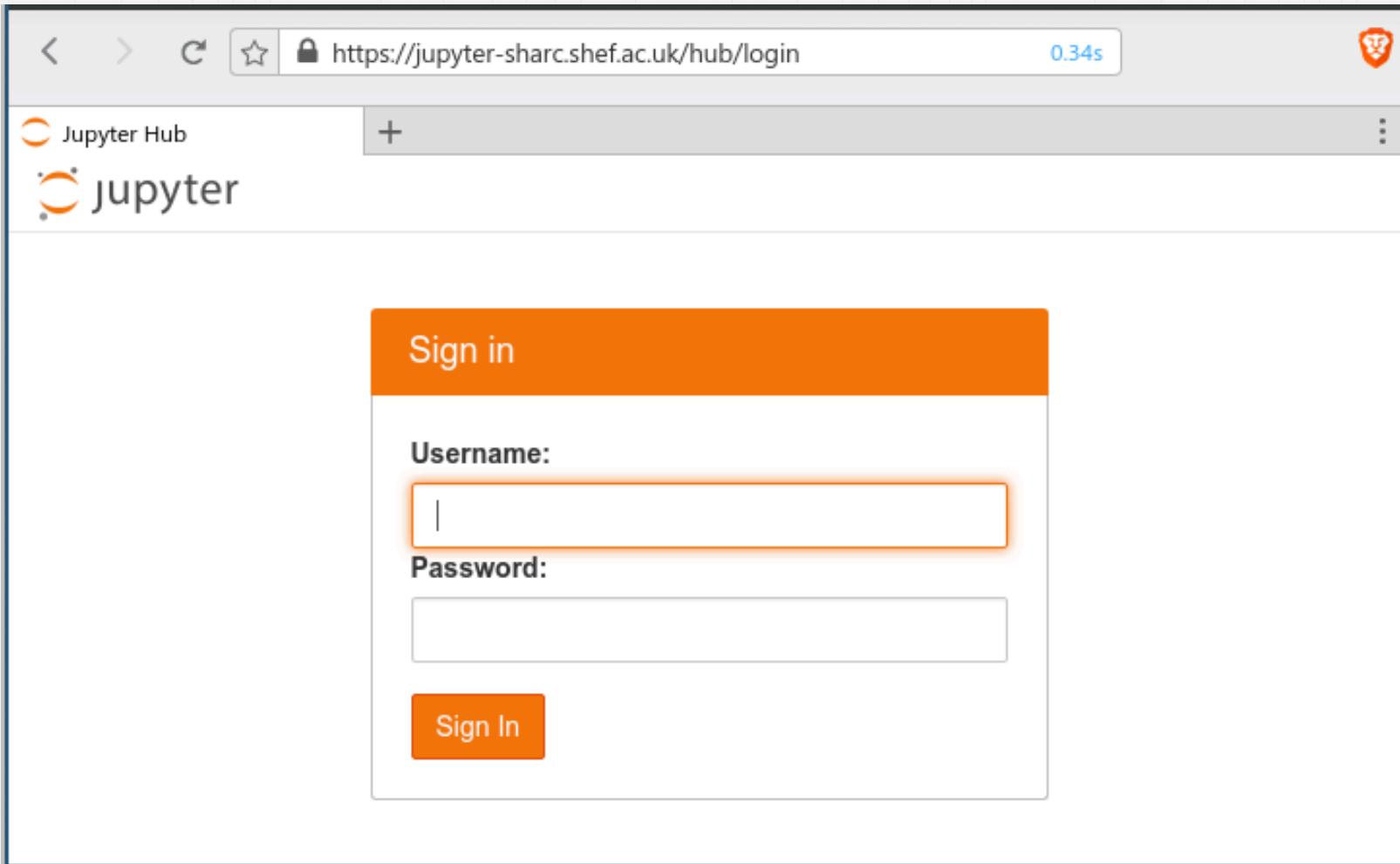
# JupyterHub

Что это?



A multi-user deployment of the notebook designed for companies,  
classrooms and research labs

# Что это?



## Преимущества

1. Юзерам больше не надо самим поднимать свои ноутбуки.
2. Централизованно все либы поставлены, настроены.
3. У всех доступ к данным где-то рядышком.

# Этап №1

# Этап №1

1. Поставим этот JupyterHub.
2. Поставим свежий Spark.
3. Сделаем так, чтобы Spark был доступен через JupyterHub.

# Установка JupyterHub

```
$ sudo adduser monty  
$ sudo passwd monty
```

```
$ npm install -g configurable-http-proxy  
$ sudo pip3 install jupyterhub
```

```
$ sudo pip3 install --upgrade notebook
```

```
$ jupyterhub
```

# Запуск при старте

```
$ sudo apt-get install supervisor  
$ sudo supervisorctl  
supervisor>  
$ cd /etc/supervisor/conf.d  
$ sudo nano jupyterhub.conf
```

```
[program:jupyterhub]  
command=/usr/local/bin/jupyterhub --config /home/ubuntu/jupyterhub/jupyterhub_config.py  
directory=/home/ubuntu/jupyterhub  
autostart=true  
autorestart=true  
startretries=3  
exitcodes=0,2  
stopsignal=TERM  
redirect_stderr=true  
stdout_logfile=/var/log/jupyter/jupyterhub.log  
stdout_logfile_maxbytes=10MB  
stdout_logfile_backups=10  
user=root
```

# Запуск при старте

```
$ sudo supervisorctl reread
```

```
$ sudo supervisorctl update
```

```
sudo supervisorctl
jupyterhub
supervisor>          RUNNING    pid 1910, uptime 4:02:27
```



# Поставим свежий Spark

```
$ wget `link to the newest version of spark`  
  
$ tar -xvf `archive`  
  
$ sudo mv spark-... /usr/hdp/{version}/  
  
$ cd /usr/hdp/{version}/  
  
$ sudo mv spark2 spark2-old  
  
$ sudo mv spark-... spark2  
  
$ cd spark2-old  
  
$ readlink conf      #copy the path from readlink  
  
$ cd ../spark2  
  
$ sudo rm -r conf  
  
$ sudo ln -s `path from readlink` conf
```

проделаем это на всех нодах

# Spark + JupyterHub

```
$ pyspark      #first of all, check if it works from console
```

```
$ cd /usr/local/share/jupyter/kernels/python3
```

```
$ sudo nano kernel.json
```

```
{  
    "argv": [  
        "python",  
        "-m",  
        "ipykernel_launcher",  
        "-f",  
        "{connection_file}"  
    ],  
    "display_name": "Python 3",  
    "language": "python",  
    "env": {  
        "SPARK_HOME": "/usr/hdp/current/spark2-client",  
        "PYTHONPATH": "$SPARK_HOME/python/:$PYTHONPATH"  
    }  
}
```

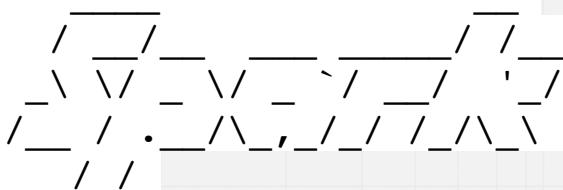
```
$ sudo supervisorctl restart jupyterhub
```

# Magic lines

```
import os
import sys
os.environ["PYSPARK_SUBMIT_ARGS"]='--packages com.databricks:spark-csv_2.10:1.2.0 pyspark-shell'
os.environ["PYSPARK_PYTHON"]='python3'
os.environ["SPARK_HOME"]='/usr/hdp/current/spark2-client'

spark_home = os.environ.get('SPARK_HOME', None)
if not spark_home:
    raise ValueError('SPARK_HOME environment variable is not set')
sys.path.insert(0, os.path.join(spark_home, 'python'))
sys.path.insert(0, os.path.join(spark_home, 'python/lib/py4j-0.10.6-src.zip'))
os.environ["PYSPARK_PYTHON"] = 'python3'
exec(open(os.path.join(spark_home, 'python/pyspark/shell.py')).read())
```

Welcome to



version 2.3.0

Using Python version 3.6.5 (default, May 3 2018 10:08:28)  
SparkSession available as 'spark'.

# Этап №2

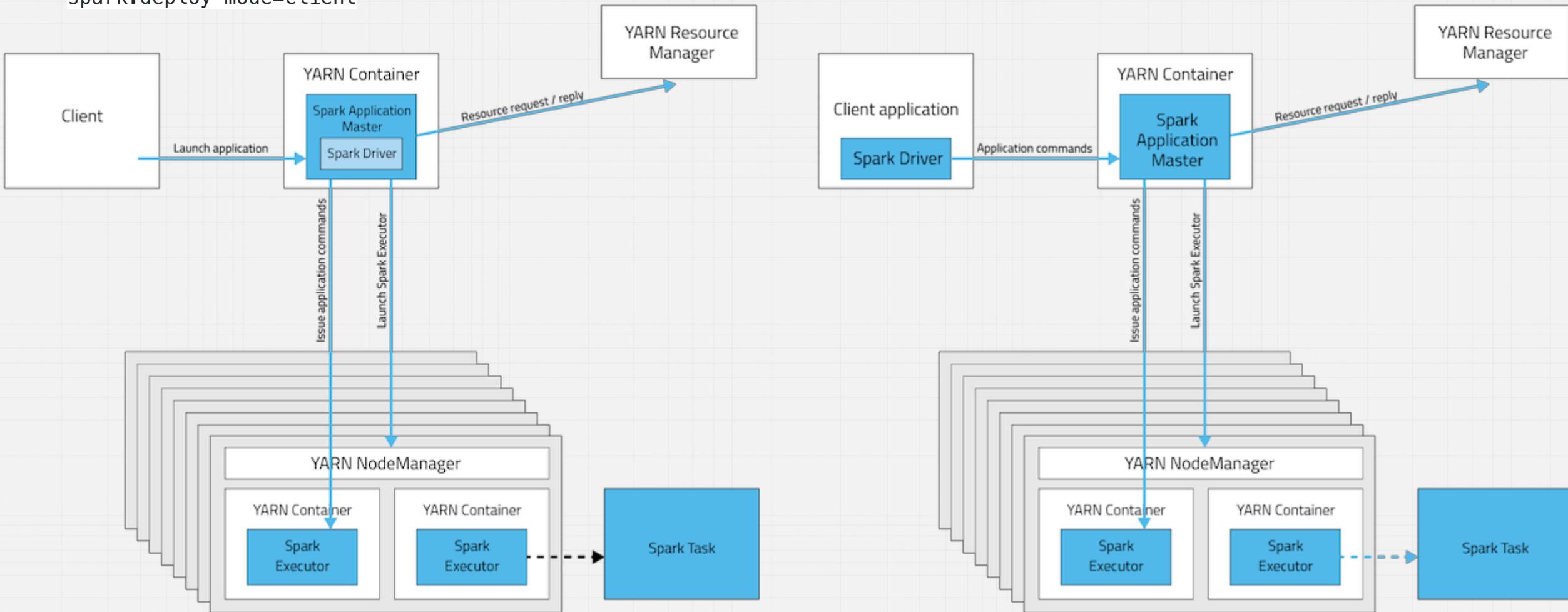
## Этап №2

1. Заведем Spark на YARN
2. Ускорим его инициализацию на ~10 секунд
3. Попробуем пару оптимизаций (kryoserializer & apache arrow)

# YARN

spark.master=yarn

spark.deploy-mode=client



# Ошибки

```
java.lang.NoClassDefFoundError: com/sun/jersey/api/client/config/ClientConfig
```

# Ошибки

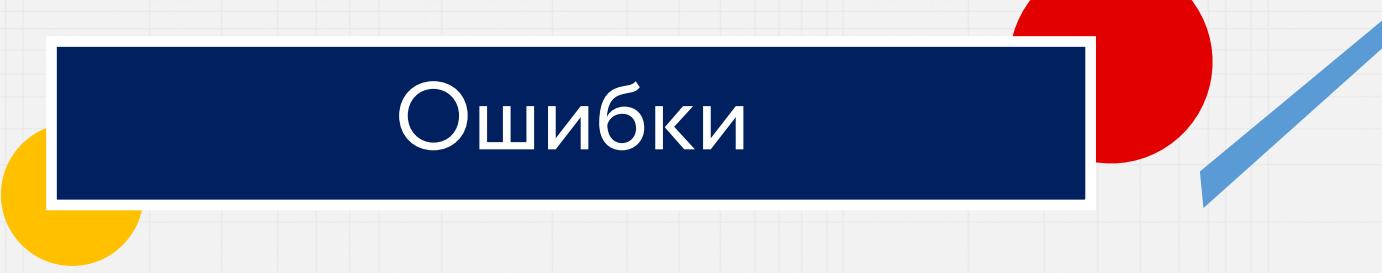
`java.lang.NoClassDefFoundError: com/sun/jersey/api/client/config/ClientConfig`

В конфиге YARN:

`yarn.timeline-service.enabled=false`



# Ошибки



`org.apache.spark.SparkException: Yarn application has already ended! It might have been killed or unable to launch application master`

# Ошибки

`org.apache.spark.SparkException: Yarn application has already ended! It might have been killed or unable to launch application master`

В конфиге YARN:

```
hdp.version=... #you can find your version inside the path /usr/hdp/
```

# Ошибки

В конфиге YARN:

```
yarn.nodemanager.aux-services=spark_shuffle,mapreduce_shuffle
```

# Ошибки

Чтобы на Hive не ругался:

```
spark.sql.catalogImplementation=in-memory
```

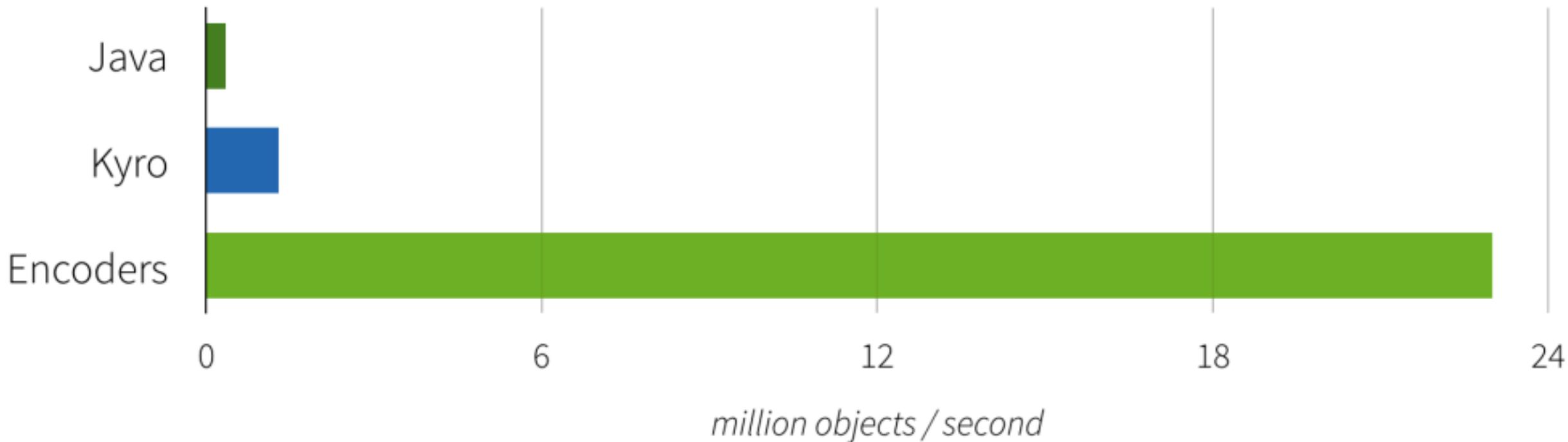
# Инициализация быстро

```
$ cd /usr/hdp/current/spark2/jars  
$ zip -r spark-archive.zip *  
$ hdfs dfs -put spark-archive.zip /tmp/ #or somewhere else
```

```
spark.yarn.archive=hdfs:///tmp/spark-archive.zip
```

# Kryo Serializer

## Serialization / Deserialization Performance



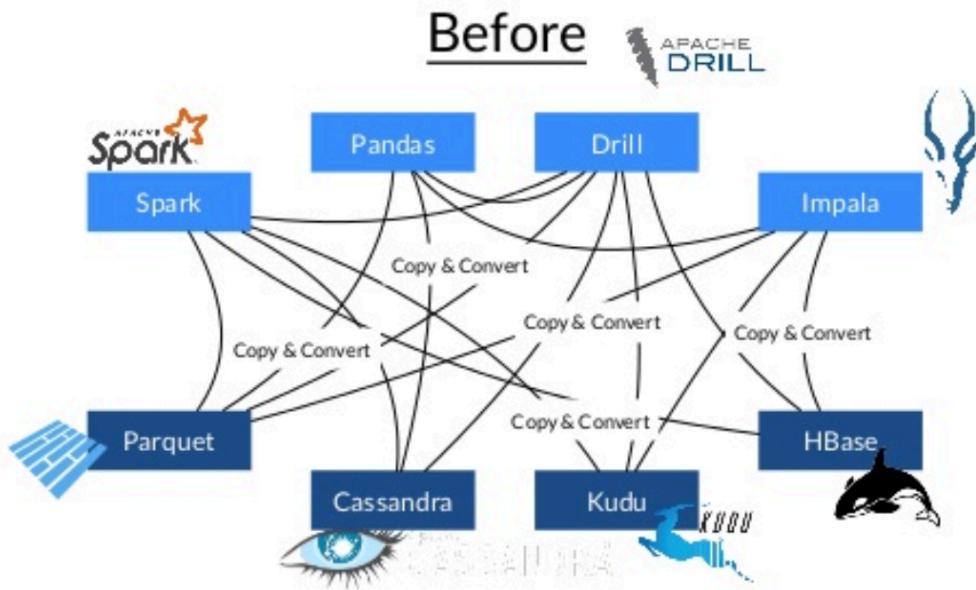
```
spark.serializer=org.apache.spark.serializer.KryoSerializer
```

```
spark.kryoserializer.buffer.max=512m
```

# Apache Arrow

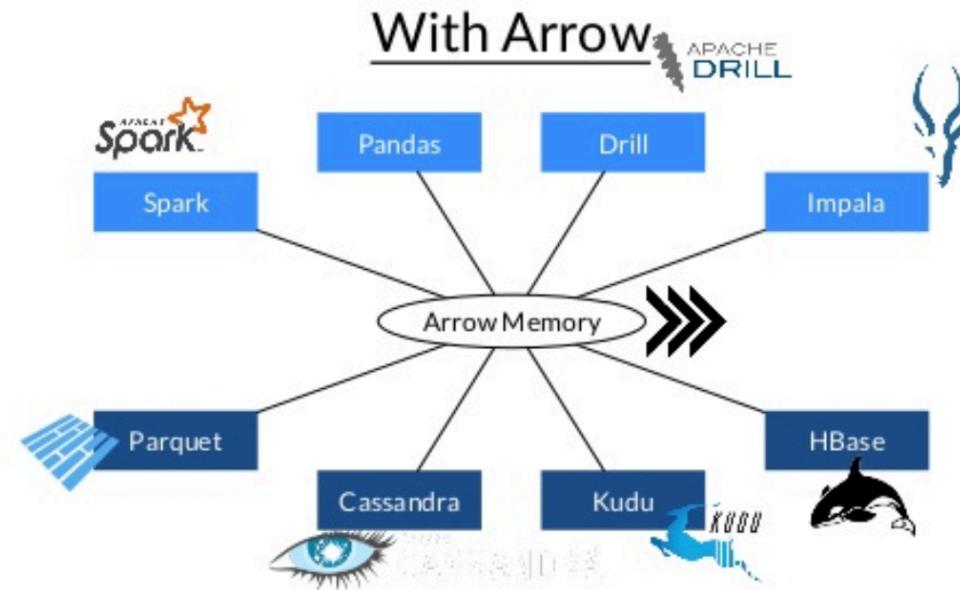
## High Performance Sharing & Interchange

### Before



- Each system has its own internal memory format
- 70-80% CPU wasted on serialization and deserialization
- Functionality duplication and unnecessary conversions

### With Arrow



- All systems utilize the same memory format
- No overhead for cross-system communication
- Projects can share functionality (eg: Parquet-to-Arrow reader)

# Arrow Benchmark

```
In [2]: %time pdf = df.toPandas()
CPU times: user 17.4 s, sys: 792 ms, total: 18.1 s
Wall time: 20.7 s

In [3]: spark.conf.set("spark.sql.execution.arrow.enabled", "true")

In [4]: %time pdf = df.toPandas()
CPU times: user 40 ms, sys: 32 ms, total: 72 ms
Wall time: 737 ms
```

<https://arrow.apache.org/blog/2017/07/26/spark-arrow/>

spark.sql.execution.arrow.enabled=true

# Arrow Benchmark

## Scalar vs Vectorized UDF

Actual Runtime is **2s** without profiling

```
8787091 function calls in 4.084 seconds
ordered by: internal time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
 20973    1.296   0.000    3.820   0.000 serializers.py:223(_batched)
2097152   0.800   0.000    2.004   0.000 worker.py:107(<lambda>)
2097152   0.761   0.000    1.204   0.000 worker.py:72(<lambda>)
2097152   0.443   0.000    0.443   0.000 <ipython-input-2-853f857cd265>:14(<lambda>)
2097152   0.214   0.000    0.214   0.000 [method 'append' of 'list' objects]
 20972   0.153   0.000    0.153   0.000 [built-in method _pickle.loads]
 20972   0.086   0.000    0.086   0.000 [built-in method _pickle.dumps]
 20972   0.046   0.000    0.230   0.000 serializers.py:148(_write_with_length)
 41944   0.045   0.000    0.045   0.000 [method 'write' of '_io.BufferedReader' objects]
 20973   0.044   0.000    0.287   0.000 serializers.py:161(_read_with_length)
 41945   0.039   0.000    0.039   0.000 [method 'read' of '_io.BufferedReader' objects]
```

20x Speed Up

```
1245 function calls (1226 primitive calls) in 0.092 seconds
Ordered by: internal time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
 3    0.013   0.004   0.013   0.004 {method 'read' of '_io.BufferedReader' objects}
 2    0.012   0.006   0.012   0.006 {method 'write' of '_io.BufferedReader' objects}
 1    0.012   0.012   0.012   0.012 [built-in method _operator.add]
 2    0.011   0.006   0.011   0.006 {method 'copy' of 'numpy.ndarray' objects}
 1    0.011   0.011   0.012   0.012 {method 'to_pandas' of 'pyarrow._table.RecordBatch' objects}
 1    0.009   0.009   0.009   0.009 {built-in method from_pandas}
 1    0.006   0.006   0.006   0.006 {method 'get_result' of 'pyarrow._io.InMemoryOutputStream' objects}
 1    0.006   0.006   0.006   0.006 {method 'to_pybytes' of 'pyarrow._io.Buffer' objects}
 1    0.005   0.005   0.005   0.005 {method 'write_batch' of 'pyarrow._io._StreamWriter' objects}
 1    0.003   0.003   0.003   0.003 internals.py:329(set)
```

# Этап №3

## Этап №3

1. Пусть Spark сам решает, сколько мне ресурсов сейчас нужно.
2. По дороге поговорим о:
  - статической аллокации и динамической
  - доп. параметрах полезных при большом количестве пользователей

# Статическая аллокация

1. Нужно самому определить эти параметры:

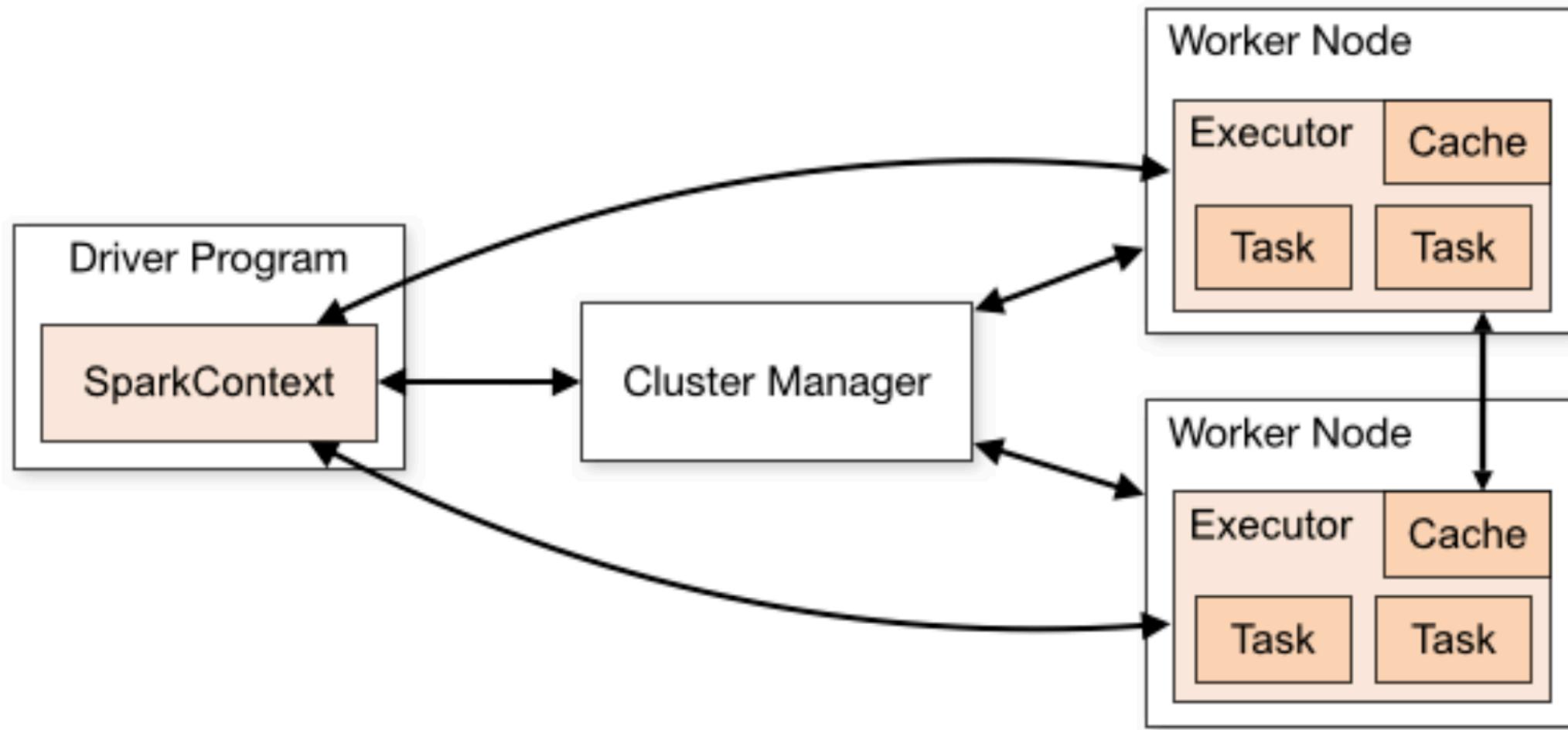
```
spark.executor.instances=
```

```
spark.executor.cores=
```

```
spark.executor.memory=
```

2. На маленьком кластере с большим количеством юзеров не очень работает.

# Статическая аллокация

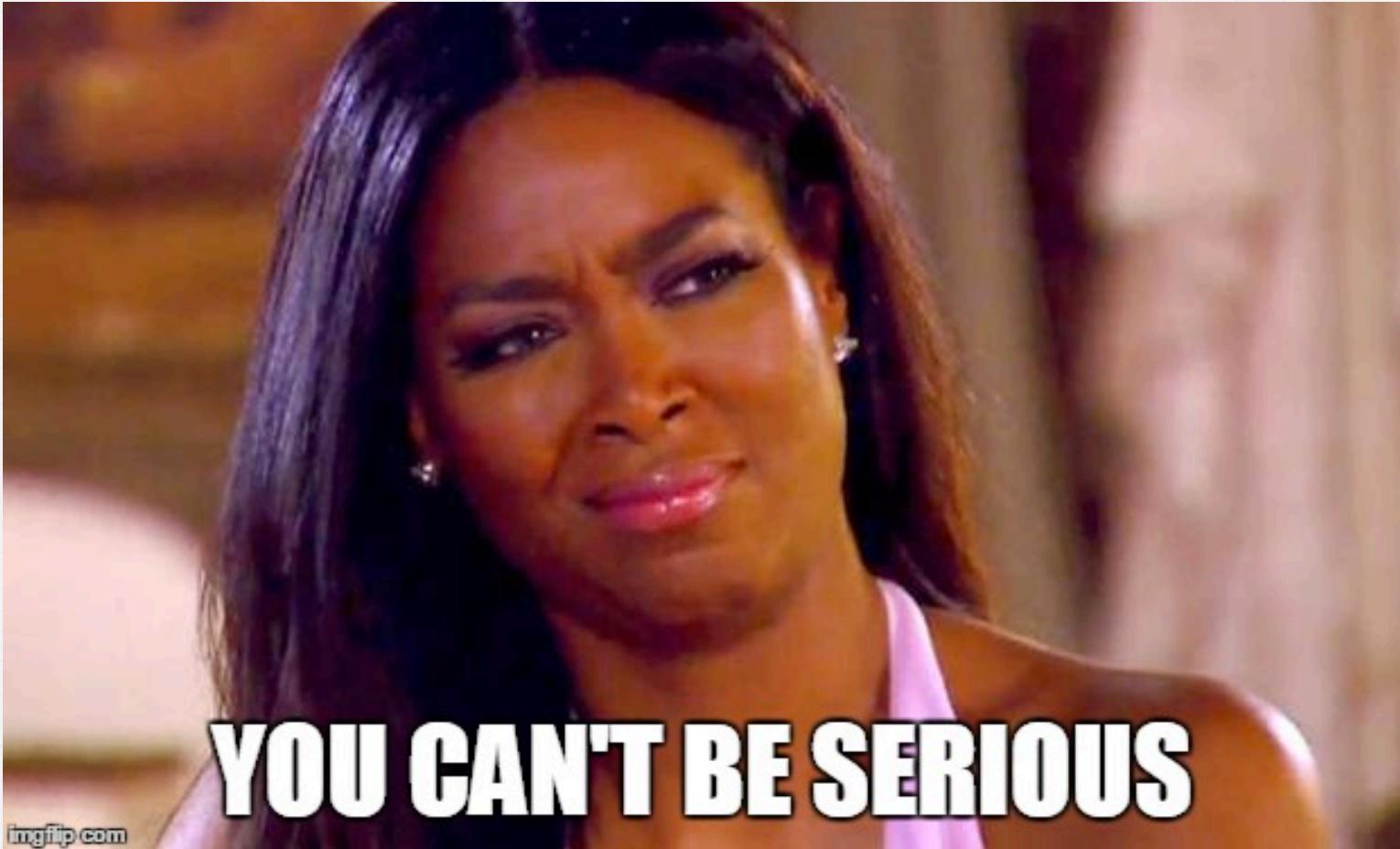


# Пример

Containers Running	Memory Used	Memory Total	Memory Reserved	Vcores Used	Vcores Total
38	171 GB	459 GB	0 B	38	114

114 ядер мы могли бы довести до 120. Равномерно поделить среди 30 юзеров. Каждому по 4 ядра. И 16 гигов оперативки.

# Пример



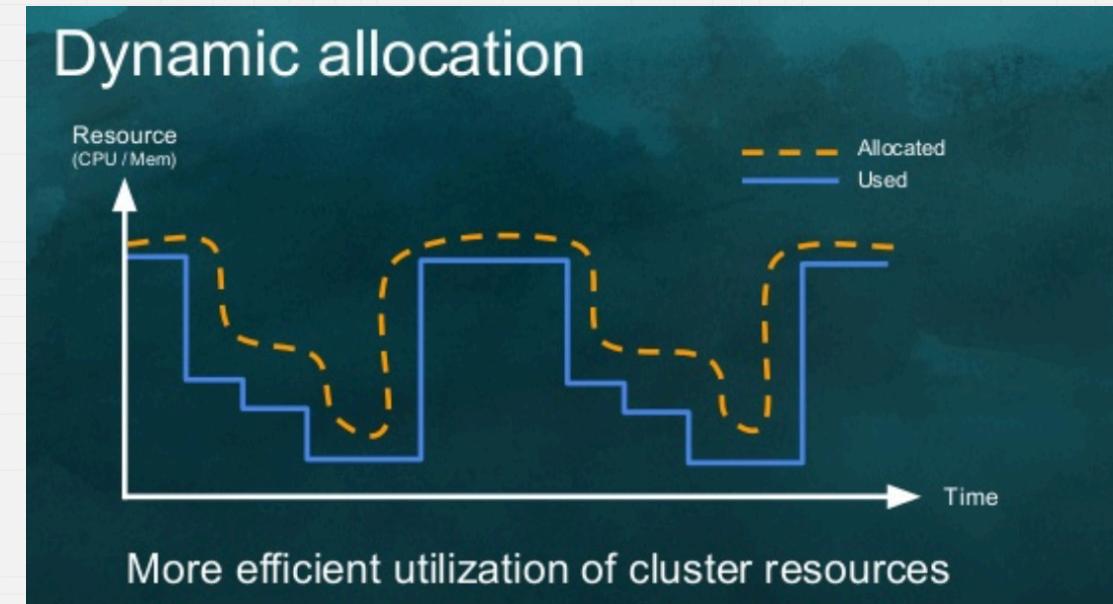
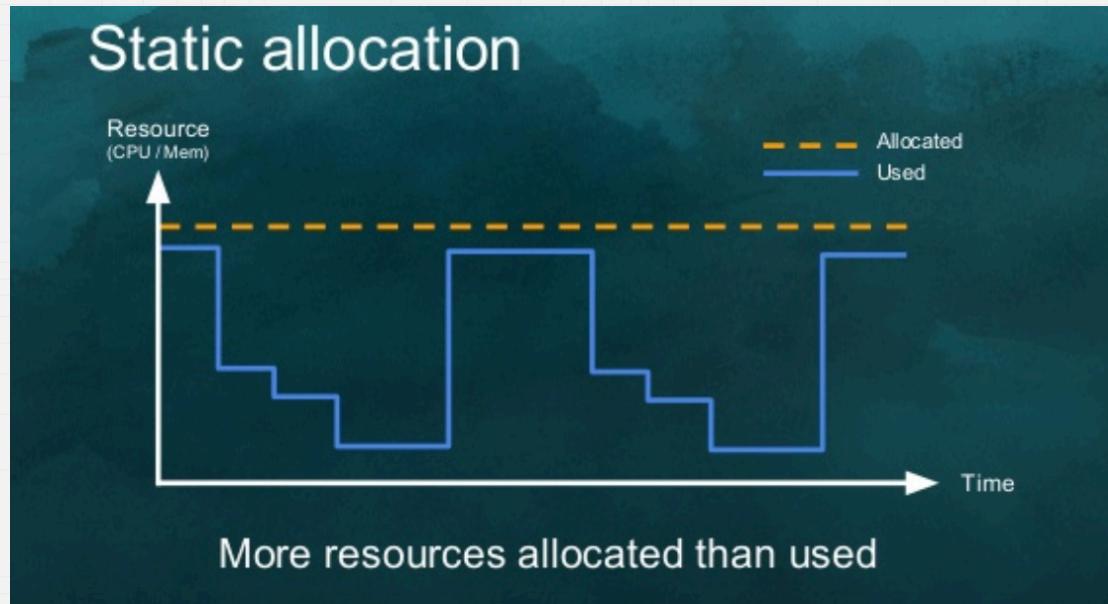
## Динамическая

Другой вариант: включить динамическую аллокацию.

Тогда Spark будет выдавать под тяжелые джобы больше ресурсов, когда кластер свободен.

```
spark.dynamicAllocation.enabled=true  
spark.shuffle.service.enabled=true
```

# Динамическая



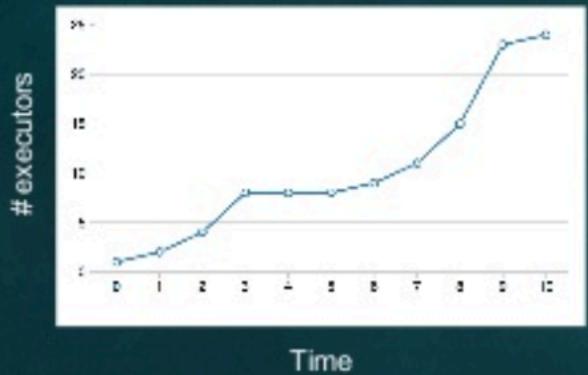
## Динамическая

## Scale up policy

*Request executors when there are pending tasks*

Scale up in multiple rounds

Exponential increase in #  
executors over time



Медленный прирост в начале (вдруг нам совсем чуть-чуть надо) и быстрая скорость роста дальше, чтобы получить много и быстро.

## Scale down policy

*Remove executors that have been idle for N seconds*



`spark.dynamicAllocation.executorIdleTimeout=120s` #by default 60s

`spark.dynamicAllocation.cachedExecutorIdleTimeout=600s` #by default infinity

`spark.dynamicAllocation.maxExecutors=14` #by default infinity

## Еще параметры

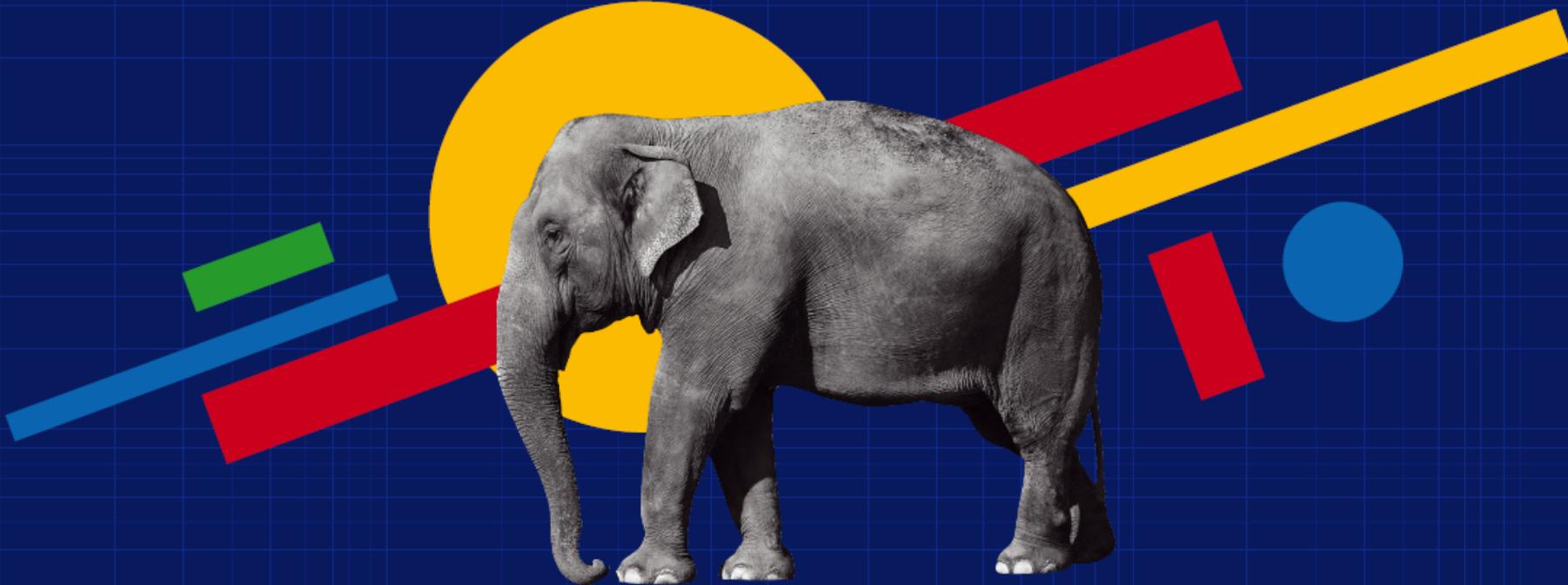
```
yarn.scheduler.capacity.maximum-am-resource-percent=1.0      #by default 0.2  
  
spark.yarn.am.memory=2G          #by default 512m  
  
spark.port.maxRetries=50        #by default 16
```

## Итог

1. Более эффективное использование ресурсов даже при большом количестве одновременных юзеров. Можно решать лабы на кластере.
2. Более равномерная загрузка кластера за счёт правильных стимулов.

# Ссылки

<https://gist.github.com/shtuder/ccc71ee761b27a0cd36f9031f434257e>



# BIG DATA IS LOVE

NEWPROLAB.COM