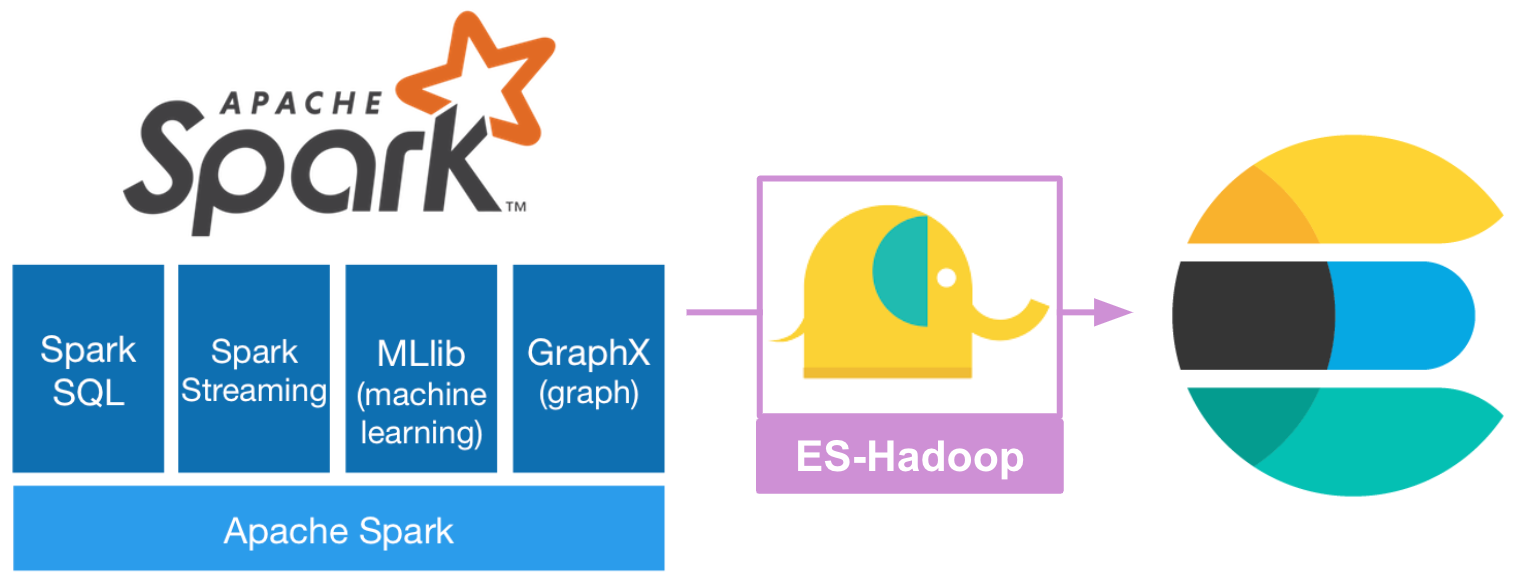
**Поиск угроз с помощью Jupyter Notebook.**

**Часть 3. Запросы Elasticsearch через Apache Spark**

В предыдущей статье я представил концепцию использования DataFrames, чтобы представить и проанализировать журналы событий безопасности в табличном формате, и показал, как это сделать с помощью библиотеки Python под названием Pandas.

В этой статье я покажу вам как использовать журналы событий безопасности непосредственно из базы данных Elasticsearch, сохранять их в DataFrame и выполнять несколько запросов через API-интерфейсы Apache Spark Python и модуль SparkSQL.

Эта статья является частью из пяти статей. Остальные четыре части можно найти по следующим ссылкам:

* Поиск угроз с помощью Jupyter Notebook. Часть 1. Ваш первый Notebook.
* Поиск угроз с помощью Jupyter Notebook. Часть 2. Базовый анализ данных с помощью Pandas.
* Поиск угроз с помощью Jupyter Notebook. Часть 4. SQL JOIN через Apache SparkSQL.
* Поиск угроз с помощью Jupyter Notebook. Часть 5. Документирование, совместное использование и запуск книг-охотников за угрозами!

**Требования**

* Предполагается, что вы читали предыдущую статью и открывали проект HELK, следуя указанным инструкциям.

Давайте рассмотрим несколько концепций Apache Spark, которые будут в этой статье.

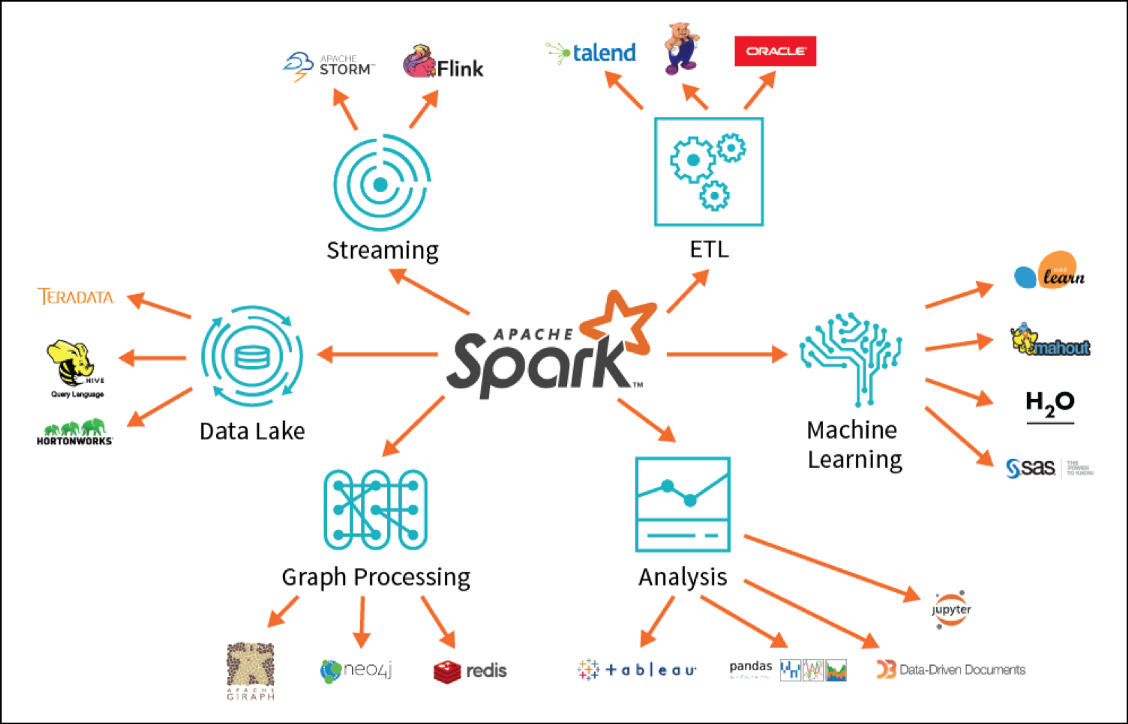
**Что такое Apache Spark?**

*Apache Spark — это унифицированный вычислительный движок и набор библиотек для параллельной обработки данных на компьютерных кластерах.*

**Единая аналитика?**

Поддерживает широкий спектр задач анализа данных на одном и том же вычислительном движке и с согласованным набором API. Spark может объединить следующие API в одно сканирование данных, например:

* простая загрузка данных,
* SQL-запросы,
* машинное обучение,
* потоковое вычисление.



**Вычислительный движок?**

* Spark обрабатывает загрузку данных из систем хранения и выполняет вычисления на них (не постоянное хранилище).
* Для перемещения данных требуется много ресурсов. Поэтому Spark фокусируется на выполнении вычислений над данными независимо от того, где они находятся.

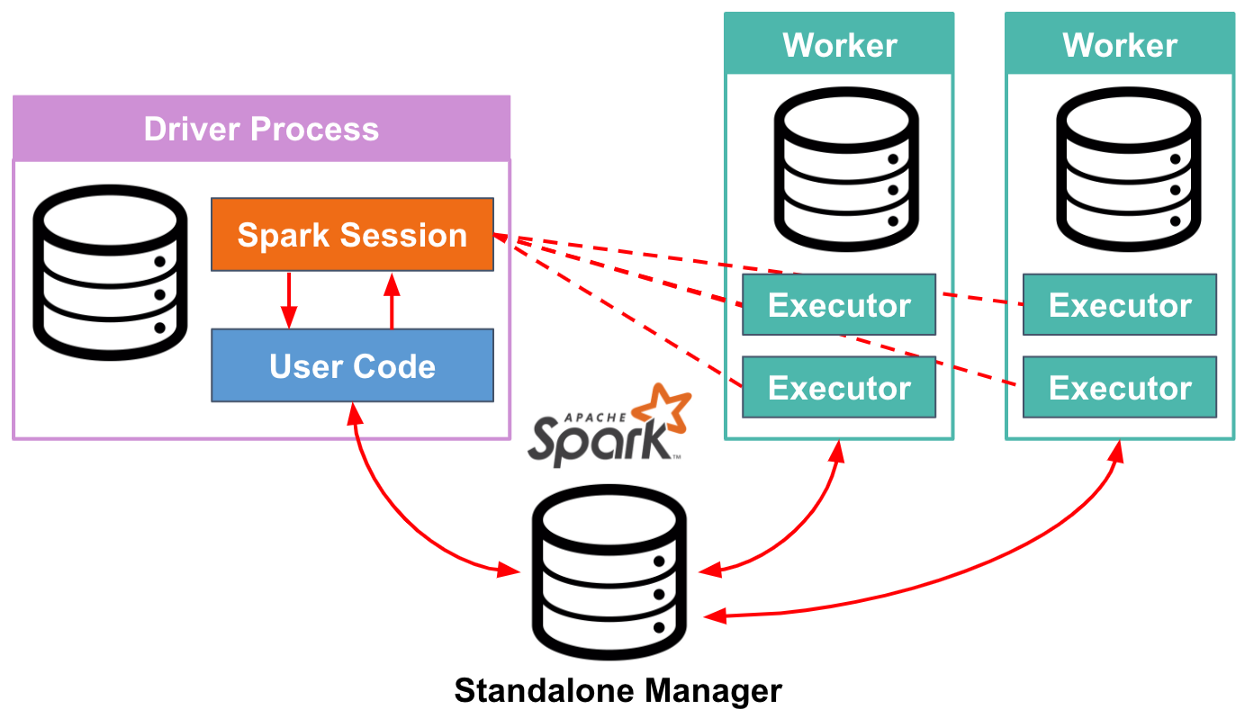
**Набор библиотек?**

Spark поддерживает следующие библиотеки:

* SQL и структурированные данные (Spark SQL);
* машинное обучение (MLlib);
* потоковая обработка (Spark Streaming и более новая структурированная потоковая передача);
* графическая аналитика (GraphX);
* внешние библиотеки доступны по адресу **spark-packages.org** (например, Graphframes).

**Базовая Архитектура Spark**

Одной из проблем анализа данных является обработка данных, которая обычно связана с доступными вычислительными ресурсами. Spark представляет концепцию параллельной обработки данных с дополнительными возможностями для координации и планирования доступных ресурсов и выполнения задач среди работников кластера.



**Процесс драйвера**

* Также называется Spark Session и запускает вашу функцию main().
* Управляет информацией о приложении Spark.
* Обрабатывает ответы на пользовательский код или ввод.
* Анализирует, распределяет и планирует работу среди исполнителей.

**Исполнители**

* Выполняют задачи, назначенные драйвером процесса.
* Выполняют и сообщают драйверу о состоянии вычисления задачи.

**Менеджер кластеров**

* Отслеживает доступные ресурсы.

**Как мне запустить Spark Code?**

* Код можно запускать через API Spark Language, например Scala, Python и т. д.
* Концепции, представленные через API языка, переводятся в код Spark и запускаются на кластере машин.

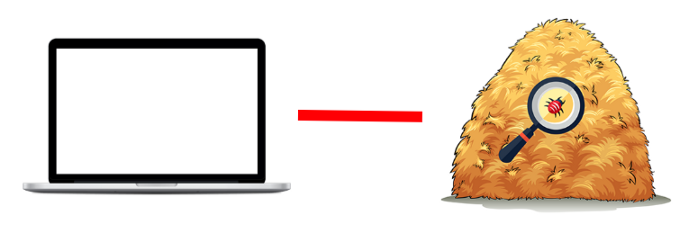
**Spark API**

Несмотря на то, что вы можете взаимодействовать с распределёнными данными и выполнять код через API Spark Language, есть два основных API Spark, благодаря которым это возможно:

* низкоуровневые неструктурированные API,
* структурированные API более высокого уровня.

**Структурированный API: Spark Dataframe**

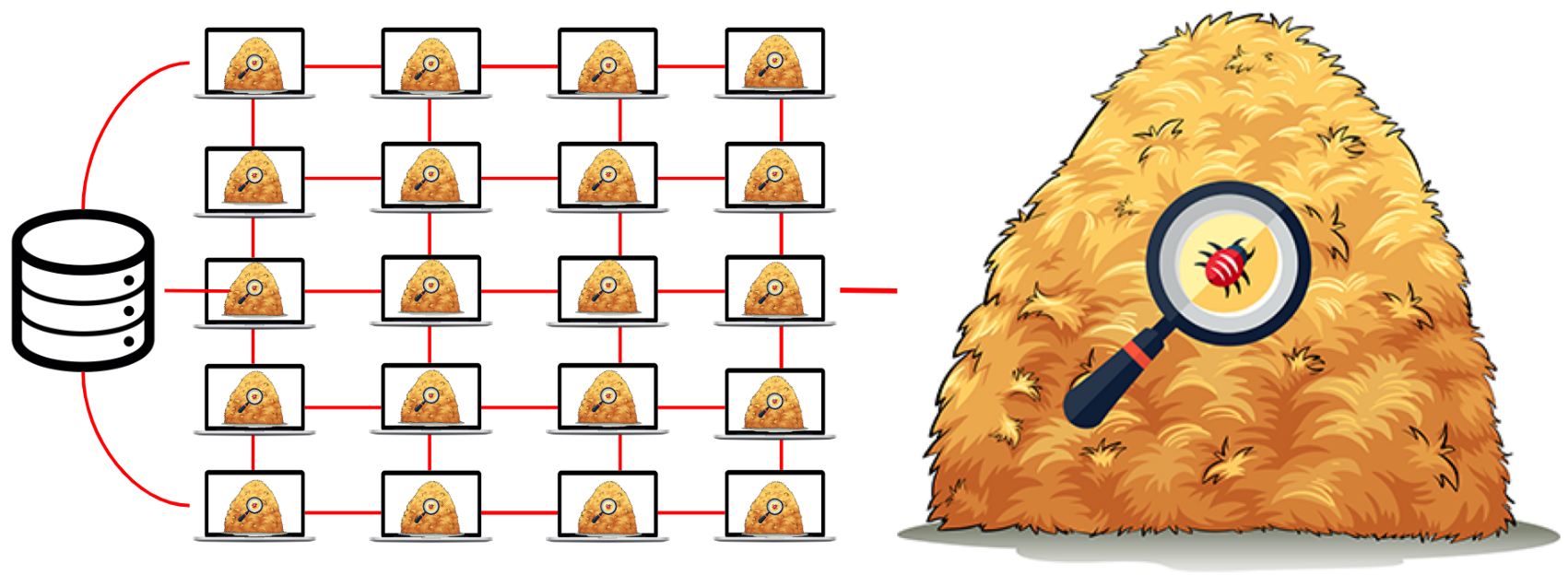
* DataFrame является наиболее распространённым структурированным API и просто организует данные в именованные столбцы и строки, как таблица в реляционной базе данных.
* Рассматривать DataFrame можно как электронную таблицу с именованными столбцами.
* DataFrame Python находится на одном компьютере в определённом месте, тогда как DataFrame Spark может существовать на нескольких компьютерах распределённым образом.



* Если DataFrame больше, чем ваш локальный жесткий диск, вы не справитесь с ним на одной машине.



* С помощью концепции Spark DataFrame можно выполнять параллельный распределённый анализ, разбивая DataFrame по частям на несколько серверов.

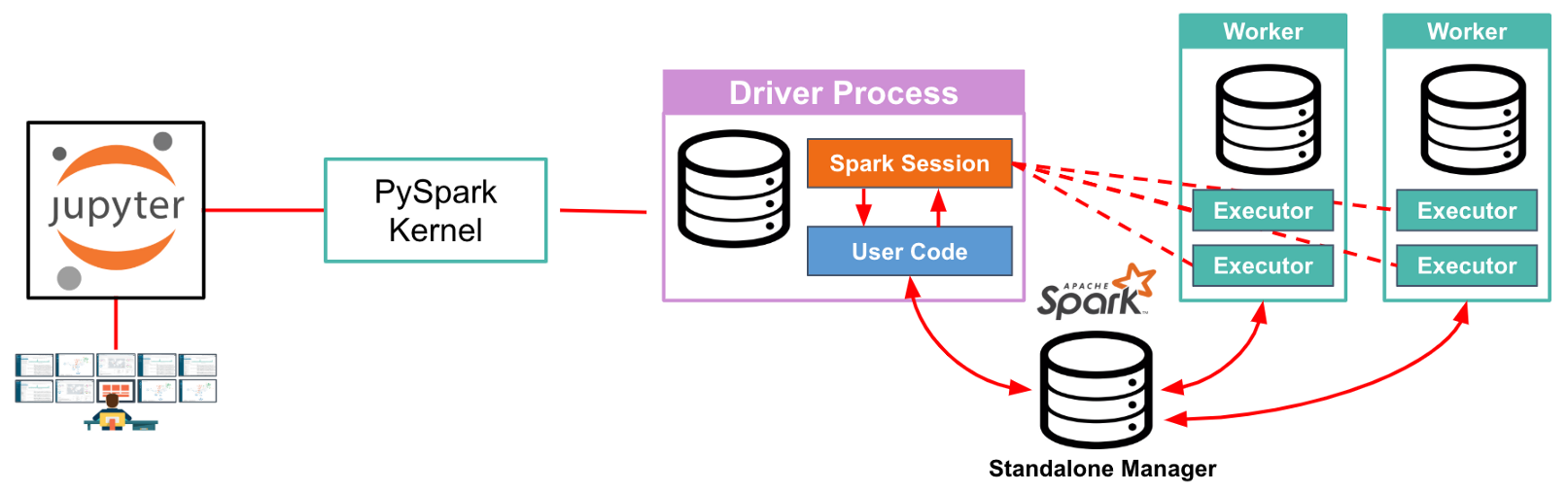


**Что такое Apache SparkSQL?**

* Это модуль Spark, использующий API-интерфейсы функционального программирования Spark для обработки реляционных структурированных данных SQL.
* Он предоставляет программную абстракцию под названием DataFrames. При запуске SparkSQL из другого языка программирования результаты возвращаются в виде Spark DataFrames. В этой статье мы будем использовать основы Pyspark для взаимодействия с DataFrames через модуль Spark SQL.

**Что такое PySpark?**

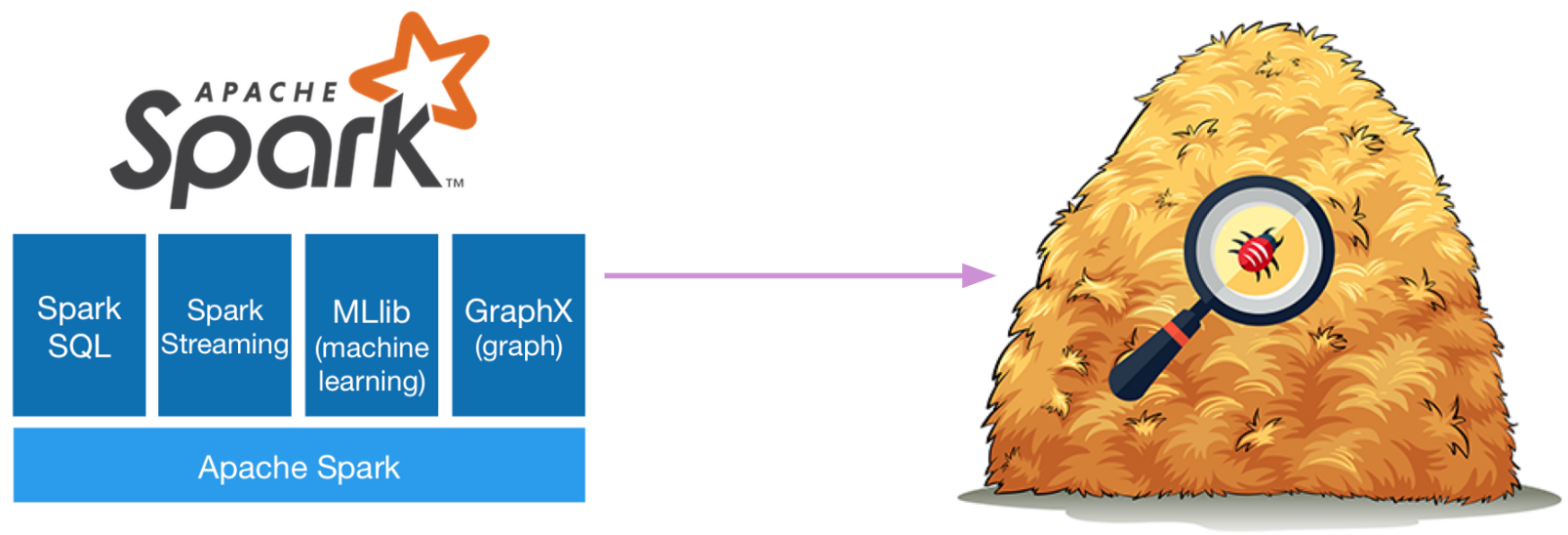
* PySpark — это API Python для Spark.
* API DataFrame доступен в Scala, Java, Python и R.
* Мы можем создать ядро ​​Jupyter, чтобы использовать API-интерфейсы PySpark и взаимодействовать с кластером Spark через блокнот. HELK уже предоставляет один.



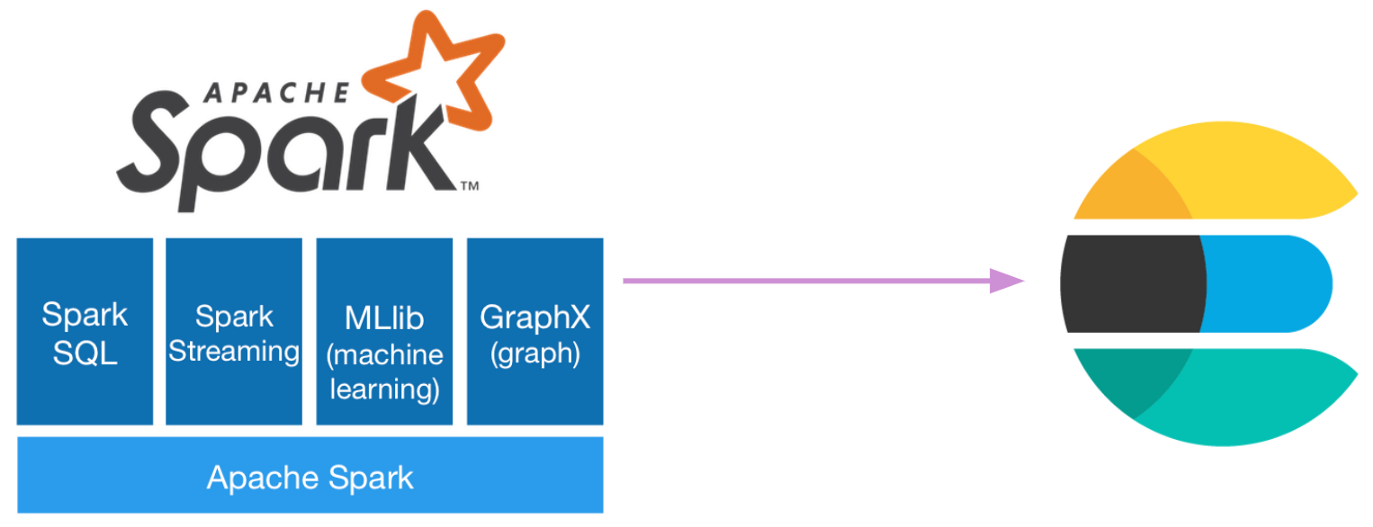
Теперь, когда вы понимаете основы Apache Spark, Spark DataFrames и API-интерфейсов Spark Language, таких как PySpark, мы прочитаем кое-какие данные и выполним несколько запросов.

**Мне нужен источник данных!**

Как упоминалось ранее, Spark фокусируется на выполнении вычислений над данными, независимо от того, где они находятся. Поэтому важно определить, где находятся данные, и как Spark может получить доступ к ним.



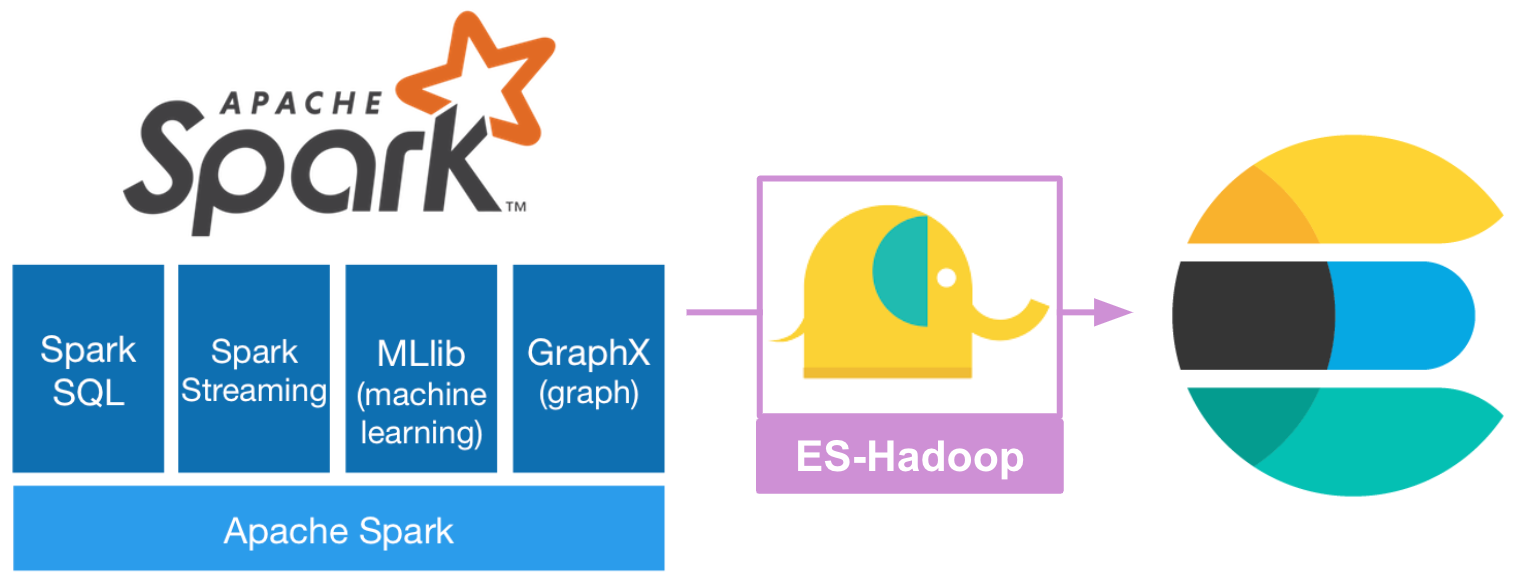
В настоящее время я храню журналы событий безопасности, собранные по сети, в базе данных Elasticsearch (ES) в рамках проекта HELK. Поэтому мне нужно было найти способ сделать запрос к моей базе данных ES через SparkSQL.



**Ввод ES-Hadoop**

К счастью, интеграция между ES и Spark Cluster осуществляется через библиотеку Elastic ES-Hadoop.

* Открытая, автономная, небольшая библиотека, которая позволяет заданиям Hadoop взаимодействовать с Elasticsearch.
* С помощью этого можно передавать данные в двух направлениях, чтобы приложения могли прозрачно использовать возможности механизма Elasticsearch.

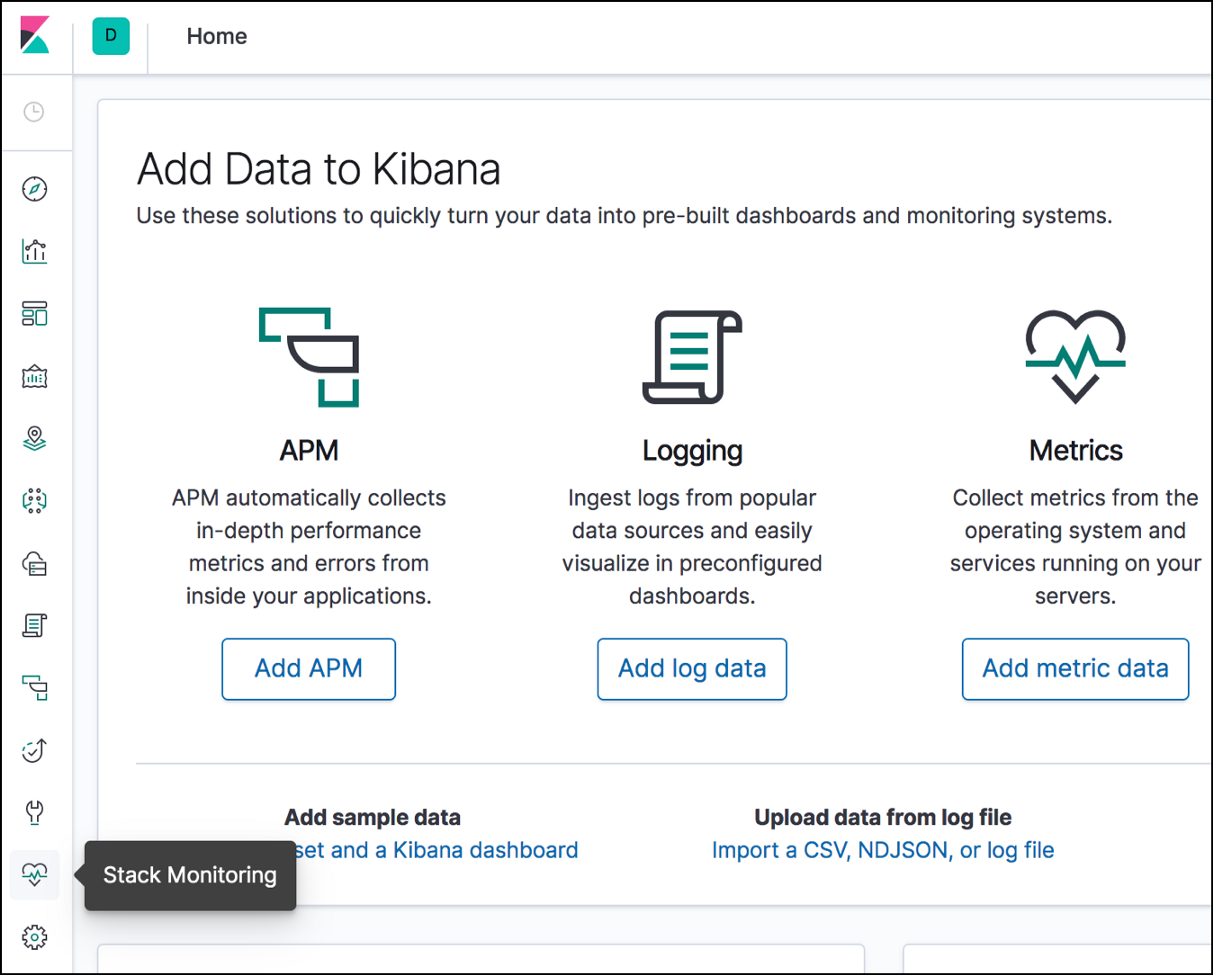


**Подготовьте Elasticsearch**

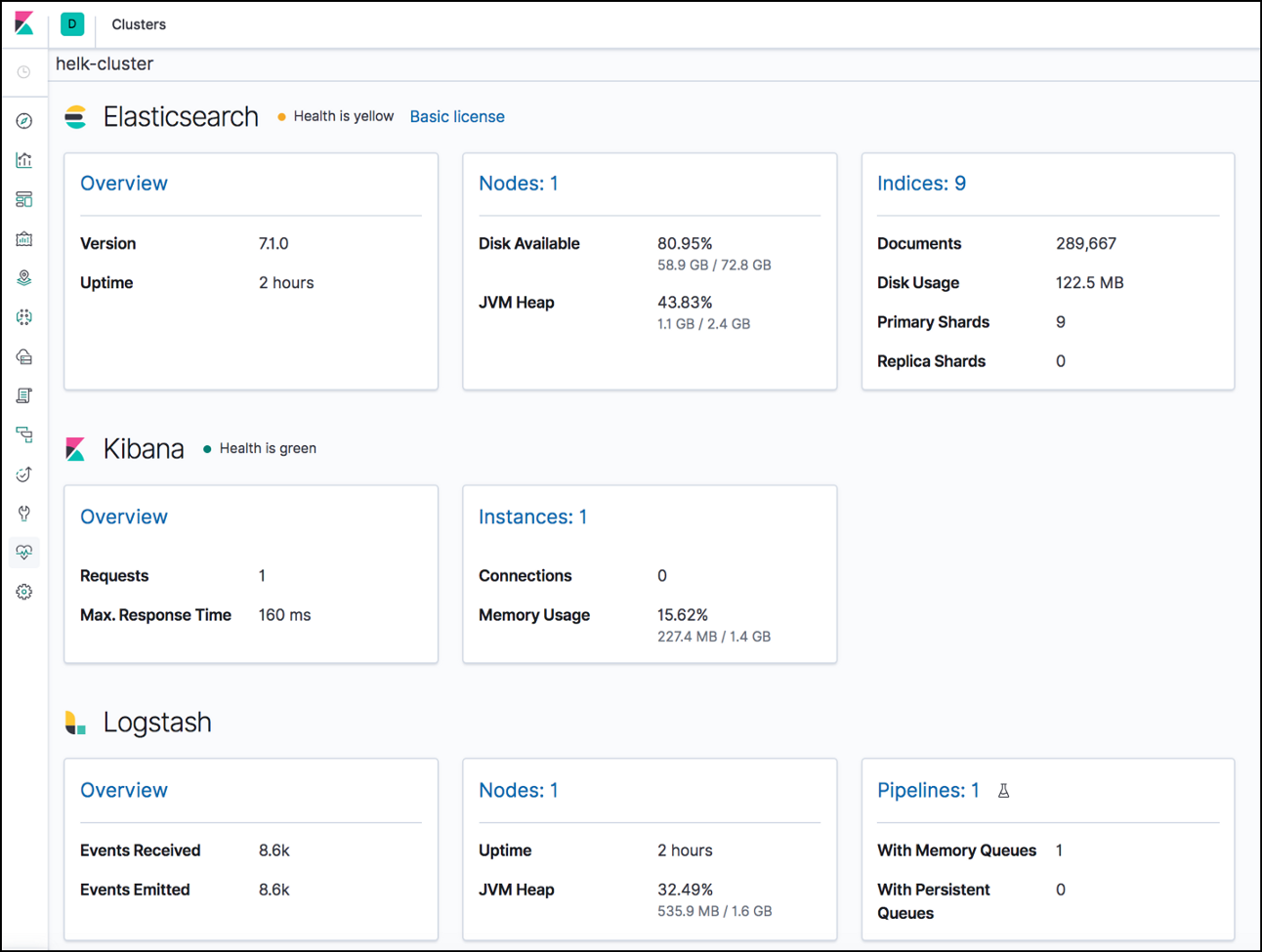
Давайте подготовим нашу среду с реальными данными внутри Elasticsearch.

**Проверьте HELK (контейнеры ELK)**

* Перейдите на IP-адрес вашего HELK и введите helk пользователя с паролем hunting.
* Нажмите на настройку stack monitoring, как показано на рисунке ниже.



* Проверьте, работает ли ваш кластер и содержит ли все отчёты пластичных контейнеров.



**Загрузите Kafkacat**

* Если вы используете систему на основе debian, убедитесь, что установили последний пакет deb Kafkacat.
* Я рекомендую Ubuntu 18.04. Вы можете проверить её версию пакета Kafkacat deb и сравнить его с последней версией репозитория Kafkacat GitHub.
* Также можно установить его из исходного кода, следуя инструкциям Quick Build.
* После установки выполните быструю проверку соединения с помощью параметра метаданных -L и укажите тему winlogbeat HELK Kafka. Эта тема должна автоматически появится при её создании в HELK.

kafkacat -b 192.168.64.138:9092 -t winlogbeat -LMetadata for winlogbeat (from broker 1: 192.168.64.138:9092/1):  
 1 brokers:  
 broker 1 at 192.168.64.138:9092  
 1 topics:  
 topic "winlogbeat" with 1 partitions:  
 partition 0, leader 1, replicas: 1, isrs: 1

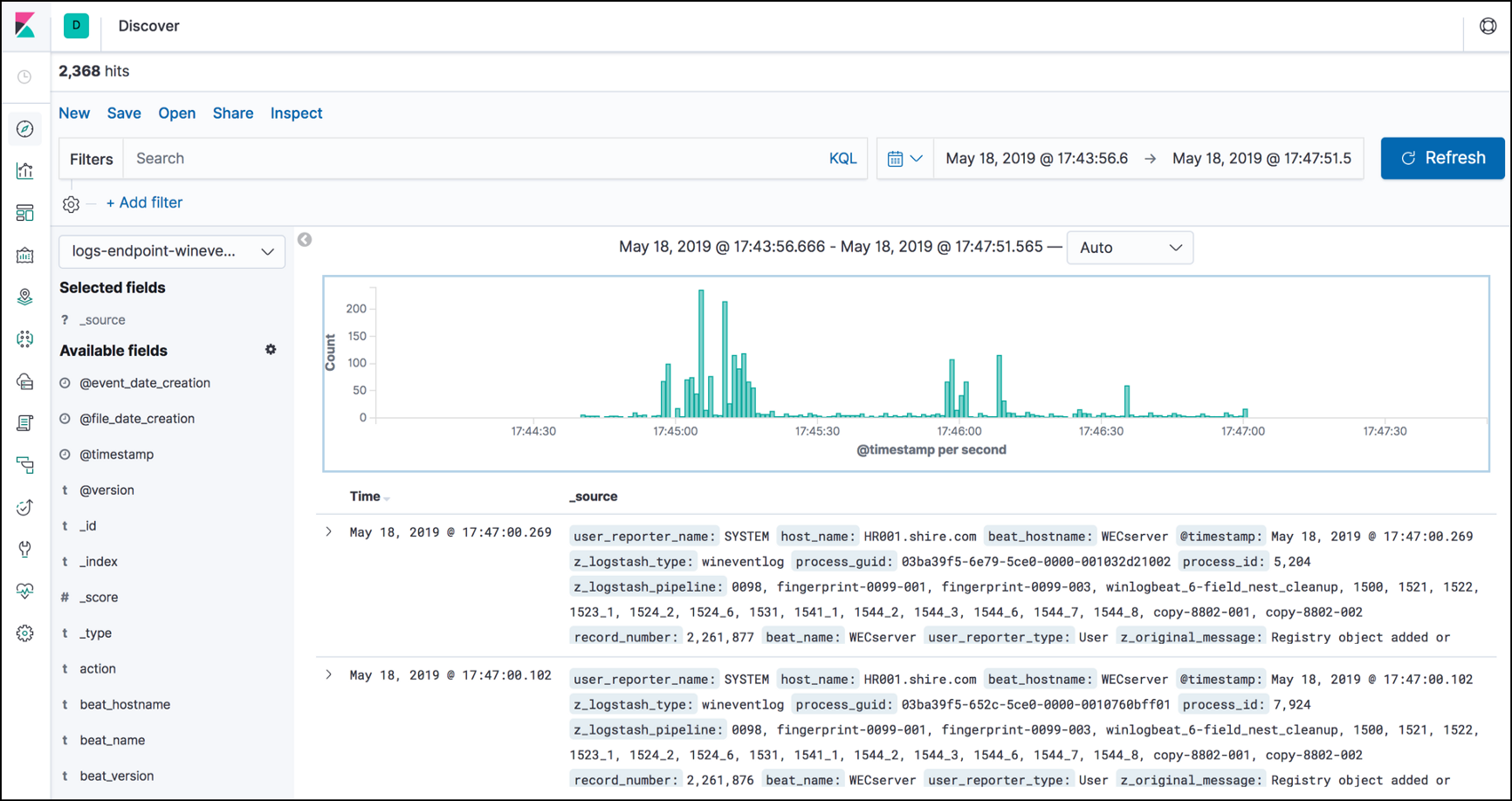
**Транзакция набора данных Мордор**

Вместо того, чтобы поддерживать целую лабораторию обнаружения или несколько компьютеров Windows на лету, мы собираемся использовать тот же набор данных Mordor empire\_invoke\_wmi, который мы использовали в предыдущей статье, и отправлять его в наш стек HELK через Kafkacat.

kafkacat -b 192.168.64.138:9092 -t winlogbeat -P -l empire\_invoke\_wmi\_2019-05-18214442.json

**Свежий взгляд на Kibana**

* Набор данных Mordor был записан 18 мая 2019 года.
* Убедитесь, что выбрано правильное временное окно для проверки данных, которые попали в базу данных Elasticsearch.

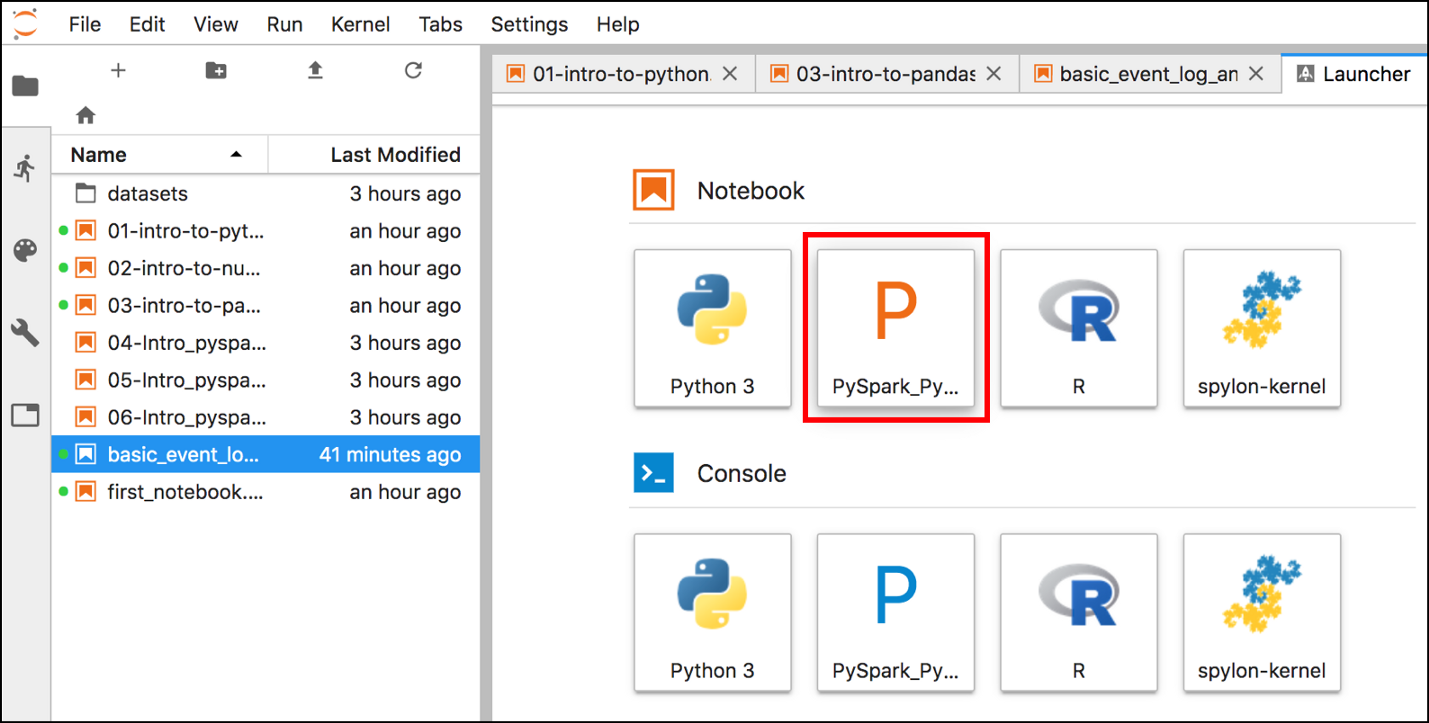


**Чтение из Elasticsearch через Apache Spark**

Мы уже готовы использовать библиотеку ES-Hadoop, чтобы Spark мог считывать, анализировать и представлять данные из Elasticsearch через его структурированные API-интерфейсы DataFrame и модуль SQL.

**Создайте новый блокнот**

* Убедитесь, что выбрали ядро ​​PySpark.



**Импортируйте SparkSession**

* Начнём с импорта класса SparkSession из модуля PySpark SQL.
* SparkSession является основной точкой входа для функций DataFrame и SQL. SparkSession пригоден для создания DataFrame, регистрации DataFrame в виде таблиц, выполнения SQL над таблицами, кэширования таблиц и даже чтения parquet-файлов.



**Создайте SparkSession**

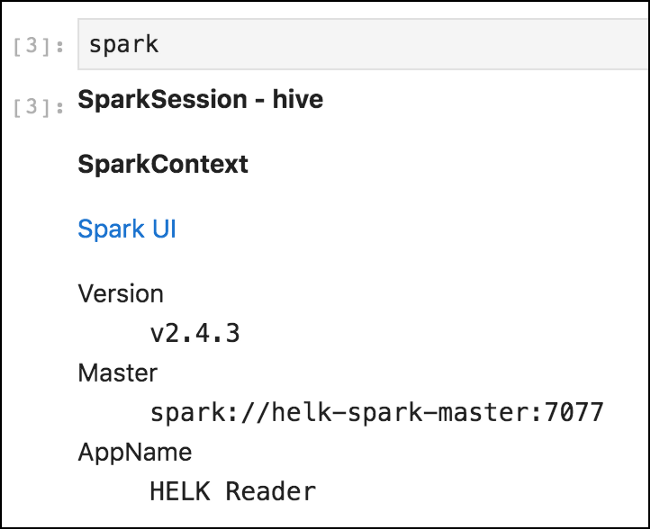
* Чтобы создать SparkSession, мы используем класс builder.
* Мы даём нашему приложению Spark имя, а также устанавливаем Spark Master в наш контейнер helk-spark-master. Обо всём этом уже позаботился HELK. Это означает, что мы собираемся использовать кластер HELK Spark для выполнения любых задач, запланированных SparkSession.

spark = SparkSession.builder \  
.appName("HELK Reader") \  
.master("spark://helk-spark-master:7077") \  
.enableHiveSupport() \  
.getOrCreate()



**Проверьте переменную Spark**

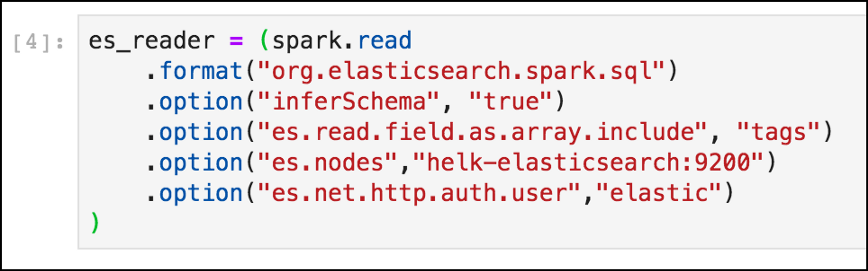
* После установки SparkSession мы сможем запустить переменную spark для проверки.



**Инициируйте Elasticsearch Reader**

* Для того, чтобы прочитать из Elasticsearch, нам нужно использовать класс DataFrameReader и метод read для чтения данных в качестве DataFrame.
* Наши основные команды только запускают DataFrame reader

es\_reader = (spark.read  
.format("org.elasticsearch.spark.sql")  
.option("inferSchema", "true")  
.option("es.read.field.as.array.include", "tags")  
.option("es.nodes","helk-elasticsearch:9200")  
.option("es.net.http.auth.user","elastic")  
)



**Загрузите данные из Elasticsearch: Sysmon Index**

* Мы можем использовать метод load для загрузки данных через считыватель DataFrame и возврата DataFrame.
* Можно указать конкретный индекс. В этом случае я выбрал индекс Sysmon.

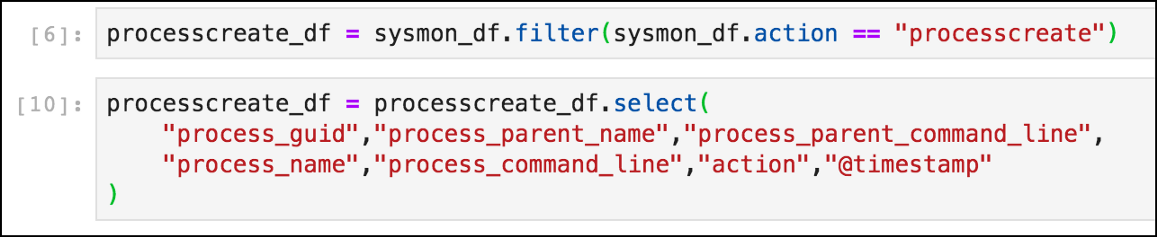
sysmon\_df = es\_reader.load("logs-endpoint-winevent-sysmon-\*/")



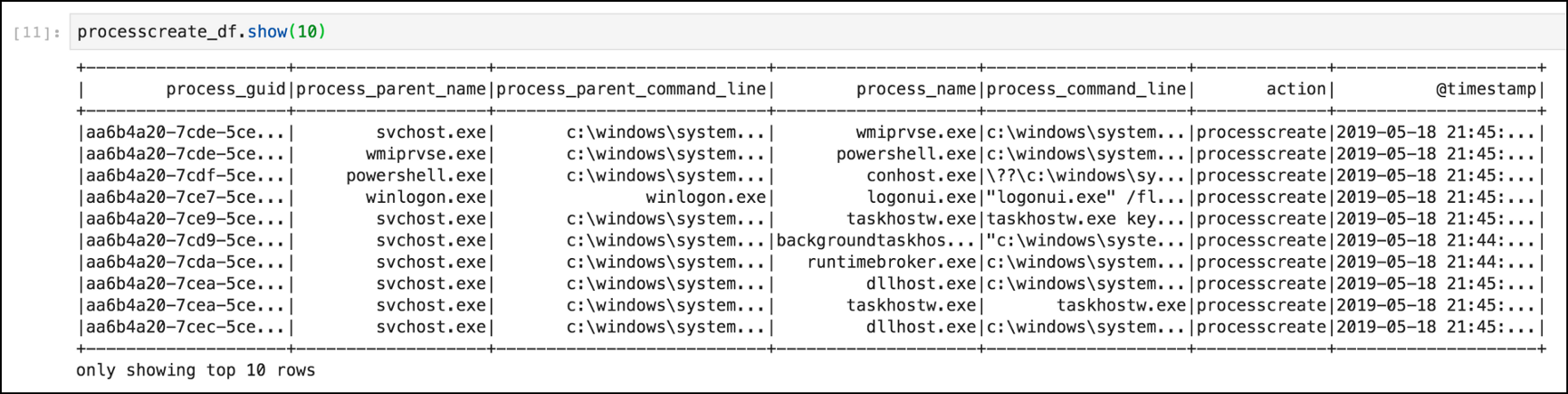
**Отфильтруйте Sysmon DataFrame**

* Метод Filter можно использовать для доступа только к событиям ProcessCreate, метод Select — для возврата определённых столбцов из Sysmon DataFrame.

processcreate\_df = sysmon\_df.filter(sysmon\_df.action == "processcreate")



**Просмотрите Sysmon ProcessCreate DataFrame**



Надеюсь, вам пока всё нравится. До сих пор мы строили основы Jupyter Notebook и использовали возможности структурированной обработки данных с помощью DataFrames для представления и анализа данных. С API Apache Spark можно сделать гораздо больше. Если вам это интересно, можете написать код и запустить его!

В следующей статье я покажу как использовать обработку реляционных данных, предоставляемую Apache SparkSQL для объединения нескольких интересных событий безопасности, которые могут быть не столь интересными или подозрительными, когда их анализируют по отдельности.

**Ссылки**

https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.SparkSession

https://spark.apache.org/docs/latest/index.html

Chambers, Bill; Zaharia, Matei. Spark: The Definitive Guide: Big Data Processing Made Simple. Kindle Edition.

https://spark.apache.org/docs/latest/api/python/pyspark.sql.html

https://databricks.com/blog/2015/02/17/introducing-dataframes-in-spark-for-large-scale-data-science.html

https://www.elastic.co/products/hadoop

https://databricks.com/glossary/what-is-spark-sql