**وزارة التعليم العالي والبحث العلمي**

**كلية شط العرب الجامعة**

**قسم علوم الحاسبات**

**Spark Architecture in big data**

**بحث مقدم الى كلية شط العرب الجامعة**

**وهو جزء من متطلبات نيل شهادة البكالوريوس في علوم الحاسبات**

**إعداد الطالب**

**محمد عبد الله نجم عبد – 97 ص**

**بأشراف**

**م.م. عمار عبد الهادي**

**1443 هـ 2022 م**

**بسم الله الرحمن الرحيم**

” قُلْ هَلْ يَسْتَوِي الَّذِينَ يَعْلَمُونَ وَالَّذِينَ لَا يَعْلَمُونَ إِنَّمَا يَتَذَكَّرُ أُولُو الْأَلْبَابِ”

«سورة الزمر: الآية 9»

الاهداء

**الى هديتي من الله، والنعمة الكبيرة التي اعيشها،**

**امي وابي، اليكما اهدي هذا البحث المتواضع،**

**عسى ان يكون صدقة جارية عني وعنكما.**

**الى اساتذتي الأعزاء**

**الذين جاهدوا واجتهدوا معنا**

**وبذلوا قصار جهدهم معنا.**

**الى أصدقاء الطرق جميعاً،**

**الوعرة والسهلة،**

**والمظلمة والمشرقة،**

**اهدي هذا البحث اذ أقدم هذا الاهداء**

**تعبيراً عن امتناني لوجودهم في حياتي.**

شكر وتقدير

**الشكر لله تعالى على فضله الذي أتاح لنا انجاز هذا العمل بفضله، فله الحمد اولاً واخراً.**

**ثم اشكر أولئك الاخيار الذين مدوا لي يد المساعدة خلال هذه الفترة وفي مقدمتهم استاذنا المشرف على البحث فضيلة الأستاذ الدكتور (باسم سهر ياسين) الذي لم يدخر جهداً في مساعدتي كما واشكر الأساتذة القائمين على قسم علوم الحاسبات في كلية شط العرب الجامعة.**

**تأييد مشرف المشروع**

**أؤيد ان مشروع التخرج المسمى: Spark Architecture in big data قد تم تحت اشرافي في كلية شط العرب الجامعة – قسم علوم الحاسبات.**

**المشرف**

**م.م. عمار عبد الهادي**

الفهرس

[الملخص 1](#_Toc102126233)

[المقدمة 1](#_Toc102126234)

[الفصل الاول 2](#_Toc102126235)

[ما هو **Apache Spark ؟** 2](#_Toc102126237)

[مكوناته 2](#_Toc102126238)

[Core Spark 4](#_Toc102126239)

[Spark SQL 4](#_Toc102126240)

[Spark Streaming 4](#_Toc102126241)

[MLlib 4](#_Toc102126242)

[GraphX 5](#_Toc102126243)

[Cluster Managers 5](#_Toc102126244)

[من يستخدم Spark ولأي غرض؟ 5](#_Toc102126245)

[مهام علوم البيانات 5](#_Toc102126246)

[تطبيقات معالجة البيانات 6](#_Toc102126247)

[تاريخ موجز Spark 7](#_Toc102126248)

[خصائص Spark 7](#_Toc102126249)

[1- السرعة 8](#_Toc102126254)

[2- سهولة الاستعمال 8](#_Toc102126255)

[3- نمطية 8](#_Toc102126256)

[4- قابلية التوسع 8](#_Toc102126257)

[الفصل الثاني 10](#_Toc102126258)

[Spark Architecture 10](#_Toc102126259)

[ماهي هيكلية Spark؟ 10](#_Toc102126260)

[1- المشغل The Driver 11](#_Toc102126261)

[1.1. تحويل برنامج المستخدم إلى مهام 11](#_Toc102126262)

[1.2. جدولة المهام على المنفذين 11](#_Toc102126263)

[2- المنفذ Executors 11](#_Toc102126264)

[إدارة المجاميع Cluster Manager 12](#_Toc102126265)

[تشغيل البرنامج Launching a Program 12](#_Toc102126266)

[الملخص 12](#_Toc102126267)

[Cluster Managers 12](#_Toc102126268)

[Standalone Cluster Manager 13](#_Toc102126269)

[Hadoop YARN 13](#_Toc102126270)

[Apache Mesos 13](#_Toc102126271)

[Amazon EC2 14](#_Toc102126272)

[سبارك: ماذا يوجد تحت RDD؟ 14](#_Toc102126273)

[هيكل سبارك Structuring Spark 15](#_Toc102126274)

[متى يستخدم RDD - When to Use RDDs 15](#_Toc102126275)

[Spark SQL والمحرك الأساسي 16](#_Toc102126276)

[الفصل الثالث 18](#_Toc102126277)

[تطبيقات في Spark 18](#_Toc102126278)

[تطبيق Anaconda 18](#_Toc102126279)

[اساسيات RDD 19](#_Toc102126280)

[امثلة تطبيقية 19](#_Toc102126281)

[1- تطبيق حساب تكرار الكلمات في الجملة 19](#_Toc102126282)

[2- تطبيق أوامر SQL DataFrames 21](#_Toc102126283)

[الفصل الرابع 24](#_Toc102126284)

[الاستنتاجات 24](#_Toc102126286)

[المصادر 25](#_Toc102126287)

فهرس الاشكال

[شكل 1 – the Spark stack 3](#_Toc102126332)

[شكل 2 - النظام البيئي للموصلات Apache Spark 9](#_Toc102126333)

[شكل 3 - مكونات تطبيق Spark الموزع 10](#_Toc102126334)

[شكل 4 - Spark SQL ومكدسها 16](#_Toc102126335)

[شكل 5 – واجهة تطبيق Anaconda 18](#_Toc102126336)

[شكل 6 – واجهة Jupyter Notebook 19](#_Toc102126337)

# الملخص

هنالك الكثير من الخوارزميات المستخدمة في لغات البرمجة الحديثة التي تحتاج الى وقت اقل لتحليل البيانات. حيث ان التعامل مع الخوارزميات بلغات البرمجة ممكن ان يعطي نتائج بوقت بطيء وذلك لكثرة البيانات. ففي هذا البحث سنلقي الضوء على Apache Spark وهيكليته ومكوناته والامكانيات التي يوفرها من ناحية السرعة في الحصول على النتائج. حيث تم تطبيق خوارزميات ممكن ان تقضي وقت طويل باستخدام لغات برمجية مثل Python, Java, Scala وغيرها. حيث تم الحصول على نتائج سريعة جداً في اغلب مجالات تحليل البيانات Big Data وData Mining وMachine Learning في الخوارزميات المستخدمة فيها مثل Clustering وClassification.

# المقدمة

بعد التطور الكبير جدا في البيانات في السنوات الأخيرة أدى ذلك الى زيادة في حجم البيانات وصعوبة التعامل معها. مما أدى الى الحاجة الى عمل أنظمة كثيرة لتبسيط البيانات وتصفيتها لسهولة التعامل معها. حيث ان Big Data والتنقيب عن البيانات Data Mining تحتاج الى الخوارزميات والطرق التي تقوم بتبسيط البيانات وتحليلها. ومن ضمن هذه الخوارزميات مثلا Clustering وClassification وRegression وغيرها الكثير. أيضا دخلت في مجال تعليم الالة Machine Learning في الذكاء الاصطناعي. مثل هكذا خوارزميات قد تحتاج الى طرق وعمليات تقلل من الوقت المستخدمة في تحليل البيانات في البرمجة المستخدمة. لان حجم البيانات في توسع هائل هذا ما قد يؤدي الى وقت كبير جدا لمعالجتها. حيث يتم استخدام الإجراءات والدوال في لغات البرمجة وبكودات حديثة ومبسطة مثل Python وغيرها من اللغات. مما أدى الى استخدام Apache Spark للتعامل مع هذه اللغات وبنفس اللغات البرمجة مثل Python, Scala, Java وغيرها. لكن بسرعة في اخراج النتائج أفضل بكثير. وذلك لاستخدامها نماذج تسهل التعامل مع العمليات الحسابية مثل MapReduce.

الفصل الاول

ما هو **Spark**

# ما هو **Apache Spark ؟**

Apache Spark عبارة عن منصة حوسبة عنقودية مصممة لتكون سريعة وذات غرض عام.

من ناحية السرعة ، يقوم Spark بتوسيع نموذج MapReduce الشهير لدعم العديد من أنواع الحسابات بكفاءة ، بما في ذلك الاستعلامات التفاعلية ومعالجة التدفق. السرعة مهمة في معالجة مجموعات البيانات الكبيرة ، لأنها تعني الفرق بين استكشاف البيانات بشكل تفاعلي وانتظار الدقائق أو الساعات. إحدى الميزات الرئيسية التي يقدمها Spark للسرعة هي القدرة على تشغيل العمليات الحسابية في الذاكرة ، ولكن النظام أيضًا أكثر كفاءة من MapReduce للتطبيقات المعقدة التي تعمل على القرص.

من ناحية العمومية ، تم تصميم Spark لتغطية مجموعة واسعة من أحمال العمل التي تتطلب مسبقًا أنظمة موزعة منفصلة ، بما في ذلك التطبيقات المجمعة ، والخوارزميات التكرارية ، والاستعلامات التفاعلية ، والتدفق. من خلال دعم أحمال العمل هذه في نفس المحرك ، تجعل Spark من السهل وغير المكلف الجمع بين أنواع المعالجة المختلفة ، والتي غالبًا ما تكون ضرورية في خطوط أنابيب تحليل بيانات الإنتاج. بالإضافة إلى ذلك ، فإنه يقلل من عبء الإدارة للحفاظ على أدوات منفصلة.

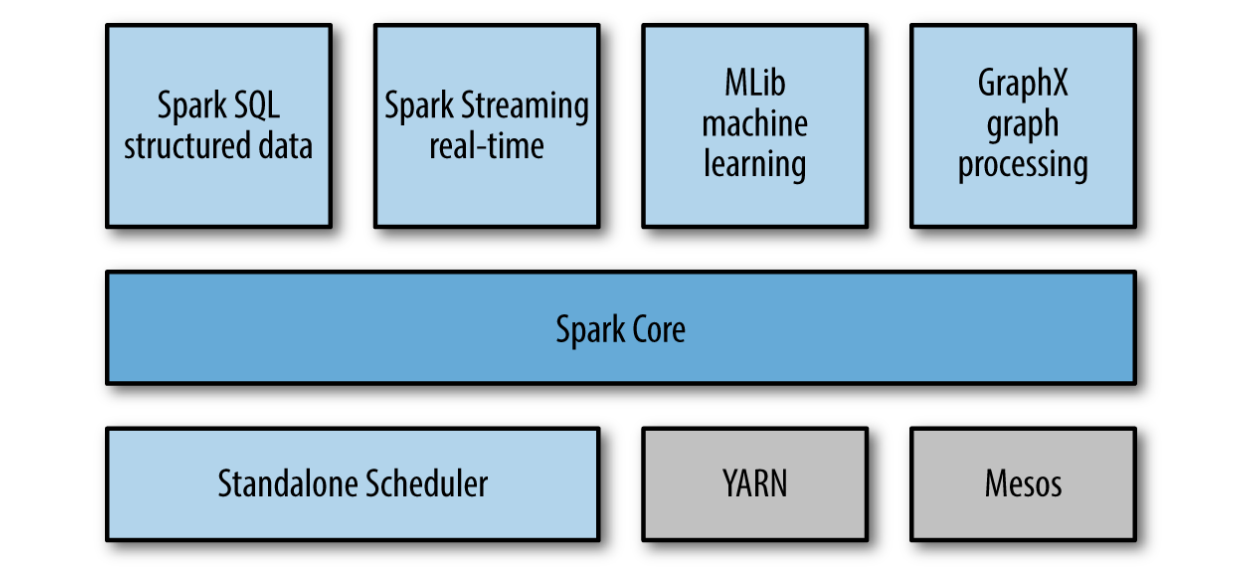
تم تصميم Spark ليكون سهل الوصول إليه بشكل كبير ، حيث يقدم واجهات برمجة تطبيقات بسيطة في Python و Java و Scala و SQL ومكتبات مدمجة غنية. كما أنه يتكامل بشكل وثيق مع أدوات البيانات الضخمة الأخرى. على وجه الخصوص ، يمكن تشغيل Spark في مجموعات Hadoop والوصول إلى أي مصدر بيانات Hadoop ، بما في ذلك Cassandra.

# مكوناته

يحتوي مشروع Spark على عدة مكونات متكاملة بشكل وثيق. يعتبر Spark في جوهره "محركًا حسابيًا" مسؤول عن جدولة وتوزيع ورصد التطبيقات التي تتكون من العديد من المهام الحسابية عبر العديد من آلات العمال ، أو مجموعة الحوسبة. نظرًا لأن المحرك الأساسي لـ Spark سريع وعامة الأغراض ، فإنه يدعم مكونات متعددة عالية المستوى متخصصة لأحمال العمل المختلفة ، مثل SQL أو التعلم الآلي. تم تصميم هذه المكونات للعمل بشكل وثيق ، مما يتيح لك دمجها مثل المكتبات في مشروع برمجي.

لفلسفة التكامل المحكم فوائد عديدة. أولاً ، تستفيد جميع المكتبات والمكونات عالية المستوى في المكدس من التحسينات في الطبقات الدنيا. على سبيل المثال ، عندما يضيف محرك Spark الأساسي تحسينًا ، فإن مكتبات SQL والتعلم الآلي تتسارع تلقائيًا أيضًا. ثانيًا ، يتم تقليل التكاليف المرتبطة بتشغيل المكدس ، لأنه بدلاً من تشغيل 5-10 أنظمة برمجية مستقلة ، تحتاج المؤسسة إلى تشغيل نظام واحد فقط. وتشمل هذه التكاليف النشر والصيانة والاختبار والدعم وغيرها. هذا يعني أيضًا أنه في كل مرة يتم فيها إضافة مكون جديد إلى Spark stack ، ستتمكن كل مؤسسة تستخدم Spark على الفور من تجربة هذا المكون الجديد. يؤدي ذلك إلى تغيير تكلفة تجربة نوع جديد من تحليل البيانات من تنزيل مشروع برمجي جديد ونشره وتعلمه إلى ترقية Spark.

أخيرًا ، تتمثل إحدى أكبر مزايا التكامل المحكم في القدرة على بناء تطبيقات تجمع بسلاسة بين نماذج المعالجة المختلفة. على سبيل المثال ، في Spark ، يمكنك كتابة تطبيق واحد يستخدم التعلم الآلي لتصنيف البيانات في الوقت الفعلي حيث يتم استيعابها من مصادر التدفق. في نفس الوقت ، يمكن للمحللين الاستعلام عن البيانات الناتجة ، أيضًا في الوقت الفعلي ، عبر SQL (على سبيل المثال ، لضم البيانات مع ملفات log‐ غير منظمة). بالإضافة إلى ذلك ، يمكن لمهندسي البيانات وعلماء البيانات الأكثر تعقيدًا الوصول إلى نفس البيانات عبر Python shell للتحليل المخصص. قد يصل الآخرون إلى البيانات في تطبيقات الدُفعات المستقلة. طوال الوقت ، يتعين على فريق تكنولوجيا المعلومات الحفاظ على نظام واحد فقط.

سنقدم هنا بإيجاز كل مكون من مكونات Spark ، كما هو موضح في الشكل 1.

شكل 1 – the Spark stack

# Core Spark

يحتوي Spark Core على الوظائف الأساسية لـ Spark ، بما في ذلك مكونات جدولة المهام وإدارة الذاكرة واستعادة الأخطاء والتفاعل مع أنظمة التخزين والمزيد. يعد Spark Core أيضًا موطنًا لواجهة برمجة التطبيقات التي تحدد مجموعات البيانات الموزعة المرنة (RDDs) ، والتي تعد تجريد البرمجة الرئيسي لشركة Spark. تمثل RDDs مجموعة من العناصر الموزعة عبر العديد من عقد الحوسبة التي يمكن معالجتها بالتوازي. يوفر Spark Core العديد من واجهات برمجة التطبيقات لبناء هذه المجموعات ومعالجتها.

# Spark SQL

Spark SQL هي حزمة Spark للعمل مع البيانات المنظمة. يسمح بالاستعلام عن البيانات عبر SQL بالإضافة إلى متغير Apache Hive من SQL - يسمى Hive Query Lan‐ guage (HQL) - ويدعم العديد من مصادر البيانات ، بما في ذلك جداول Hive و Parquet و JSON. بالإضافة إلى توفير واجهة SQL لـ Spark ، يسمح Spark SQL للمطورين بدمج استعلامات SQL مع عمليات التلاعب بالبيانات البرمجية التي تدعمها RDDs في Python و Java و Scala ، كل ذلك ضمن تطبيق واحد ، وبالتالي دمج SQL مع التحليلات المعقدة. هذا التكامل الوثيق مع بيئة الحوسبة الغنية التي توفرها Spark يجعل Spark SQL تختلف عن أي أداة أخرى لتخزين البيانات مفتوحة المصدر. تمت إضافة Spark SQL إلى Spark في الإصدار 1.0.

كان Shark مشروعًا قديمًا لـ SQL-on-Spark من جامعة كاليفورنيا ، Berke‐ ley ، قام بتعديل Apache Hive ليتم تشغيله على Spark. تم استبداله الآن بـ Spark SQL لتوفير تكامل أفضل مع محرك Spark وواجهات برمجة تطبيقات اللغة.

# Spark Streaming

Spark Streaming هو مكون Spark يتيح معالجة التدفقات الحية للبيانات. تتضمن أمثلة تدفقات البيانات ملفات السجل التي تم إنشاؤها بواسطة خوادم الويب الخاصة بالإنتاج ، أو قوائم انتظار الرسائل التي تحتوي على تحديثات الحالة التي ينشرها مستخدمو خدمة الويب. يوفر Spark Streaming واجهة برمجة تطبيقات لمعالجة تدفقات البيانات التي تتطابق بشكل وثيق مع واجهة برمجة تطبيقات RDD الخاصة بشركة Spark Core ، مما يسهل على المبرمجين تعلم المشروع والتنقل بين التطبيقات التي تعالج البيانات المخزنة في الذاكرة أو على القرص أو الوصول في الوقت الفعلي. تم تصميم Spark Streaming أسفل واجهة برمجة التطبيقات (API) الخاصة به لتوفير نفس درجة تحمل الخطأ والإنتاجية وقابلية التوسع مثل Spark Core.

# MLlib

يأتي Spark مع مكتبة تحتوي على وظائف التعلم الآلي الشائعة (ML) ، والتي تسمى MLlib. يوفر MLlib أنواعًا متعددة من خوارزميات التعلم الآلي ، بما في ذلك التصنيف والانحدار والتجميع والتصفية التعاونية ، بالإضافة إلى وظائف الدعم مثل تقييم النموذج واستيراد البيانات. كما يوفر بعض أساسيات ML ذات المستوى المنخفض ، بما في ذلك خوارزمية تحسين النسب العامة للتدرج. تم تصميم كل هذه الأساليب لتوسيع نطاقها عبر الكتلة.

# GraphX

GraphX هي مكتبة لمعالجة الرسوم البيانية (على سبيل المثال ، الرسم البياني لأصدقاء الشبكة الاجتماعية) وإجراء عمليات حسابية موازية للرسم البياني. مثل Spark Streaming و Spark SQL ، تعمل GraphX على توسيع واجهة Spark RDD API ، مما يسمح لنا بإنشاء رسم بياني موجه بخصائص arbi‐ trary مرتبطة بكل قمة وحافة. توفر GraphX أيضًا عوامل تشغيل مختلفة لمعالجة الرسوم البيانية (على سبيل المثال ، الرسم البياني الفرعي و mapVertices) ومكتبة من خوارزميات الرسم البياني الشائعة (على سبيل المثال ، PageRank وعدّ المثلث).

# Cluster Managers

تحت الغطاء ، تم تصميم Spark لتوسيع نطاق الحساب بكفاءة من واحد إلى عدة آلاف من عقد الحوسبة. لتحقيق ذلك مع زيادة المرونة إلى أقصى حد ، يمكن لـ Spark تشغيل مجموعة متنوعة من مديري المجموعات ، بما في ذلك Hadoop YARN و Apache Mesos ومدير مجموعة بسيط مضمن في Spark نفسه يسمى Standalone Scheduler. إذا كنت تقوم فقط بتثبيت Spark على مجموعة فارغة من الأجهزة ، فإن Standalone Scheduler يوفر طريقة سهلة للبدء ؛ إذا كان لديك بالفعل مجموعة Hadoop YARN أو Mesos ، ومع ذلك ، فإن دعم Spark لمديري المجموعات يسمح بتشغيل تطبيقاتك عليهم أيضًا.

# من يستخدم Spark ولأي غرض؟

نظرًا لأن Spark عبارة عن إطار عمل للأغراض العامة للحوسبة العنقودية ، يتم استخدامه لمجموعة متنوعة من التطبيقات. حددنا في المقدمة مجموعتين من القراء يستهدفهما هذا الكتاب: علماء البيانات والمهندسون. دعونا نلقي نظرة فاحصة على كل مجموعة وكيف تستخدم Spark. ليس من المستغرب أن تختلف حالات الاستخدام النموذجية بين الحالتين ، ولكن يمكننا تصنيفها تقريبًا إلى فئتين ، علم البيانات وتطبيقات البيانات.

بالطبع ، هذه أنظمة وأنماط استخدام غير دقيقة ، والعديد من الأشخاص لديهم مهارات من كليهما ، ويلعبون أحيانًا دور عالم البيانات المحقق ، ثم "تغيير القبعات" وكتابة تطبيق معالجة بيانات متشدد. بخلاف ذلك ، قد يكون من المفيد النظر في المجموعتين وحالات الاستخدام الخاصة بهما بشكل منفصل.

# مهام علوم البيانات

يركز علم البيانات ، وهو تخصص ظهر خلال السنوات القليلة الماضية ، على تحليل البيانات. على الرغم من عدم وجود تعريف قياسي ، فإن عالم البيانات هو شخص مهمته الرئيسية تحليل البيانات ونمذجتها. قد يكون لعلماء البيانات خبرة في SQL والإحصاءات والنمذجة التنبؤية (التعلم الآلي) والبرمجة ، عادةً في Python أو Matlab أو R. يمتلك علماء البيانات أيضًا خبرة في التقنيات اللازمة لتحويل البيانات إلى تنسيقات يمكن تحليلها للحصول على رؤى (يشار إليها أحيانًا باسم الجدل حول البيانات).

يستخدم علماء البيانات مهاراتهم في تحليل البيانات بهدف الإجابة عن سؤال أو اكتشاف الرؤى. في كثير من الأحيان ، يتضمن سير العمل الخاص بهم تحليلًا مخصصًا ، لذلك يستخدمون أصداف تفاعلية (مقابل إنشاء تطبيقات معقدة) تتيح لهم رؤية نتائج الاستعلامات ومقتطفات التعليمات البرمجية في أقل وقت ممكن. تتألق سرعة Spark وواجهات برمجة التطبيقات البسيطة لهذا الغرض ، وتعني مكتباتها المدمجة أن العديد من الخوارزميات متاحة خارج الصندوق.

يدعم Spark المهام المختلفة لعلوم البيانات بعدد من المكونات. تسهل Spark shell إجراء تحليل تفاعلي للبيانات باستخدام Python أو Scala. يحتوي Spark SQL أيضًا على غلاف SQL منفصل يمكن استخدامه للقيام باستكشاف البيانات باستخدام SQL ، أو يمكن استخدام Spark SQL كجزء من برنامج Spark العادي أو في Spark shell. يتم دعم التعلم الآلي وتحليل البيانات من خلال مكتبات MLLib. بالإضافة إلى ذلك ، هناك دعم للدعوة إلى البرامج الخارجية في Matlab أو R. Spark تمكن علماء البيانات من معالجة المشكلات ذات أحجام البيانات الأكبر مما كان بإمكانهم من قبل باستخدام أدوات مثل R أو Pandas.

في بعض الأحيان ، بعد مرحلة الاستكشاف الأولية ، سيتم "إنتاج" عمل عالم البيانات أو توسيعه ، وتقويته (أي جعله يتحمل الأخطاء) ، وضبطه ليصبح تطبيق معالجة بيانات الإنتاج ، والذي يعد في حد ذاته أحد مكونات تطبيقات الأعمال. على سبيل المثال ، قد يؤدي التحقيق الأولي لعالم البيانات إلى إنشاء نظام توصية الإنتاج الذي يتم دمجه في تطبيق ويب ويستخدم لإنشاء اقتراحات المنتج للمستخدمين. غالبًا ما يكون شخص أو فريق مختلف هو الذي يقود عملية تحويل عمل علماء البيانات إلى منتجات ، وغالبًا ما يكون هذا الشخص مهندسًا.

# تطبيقات معالجة البيانات

يمكن وصف حالة الاستخدام الرئيسية الأخرى لـ Spark في سياق المهندس per‐ sona. لأغراضنا هنا ، نفكر في المهندسين كفئة كبيرة من مطوري البرامج الذين يستخدمون Spark لبناء تطبيقات معالجة بيانات الإنتاج. عادة ما يكون لدى هؤلاء المطورين فهم لمبادئ هندسة البرمجيات ، مثل التغليف وتصميم الواجهة والبرمجة الموجهة للكائنات. هم حاصلون على شهادة جامعية في علوم الكمبيوتر. يستخدمون مهاراتهم الهندسية لتصميم وبناء أنظمة البرمجيات التي تنفذ حالة استخدام الأعمال.

بالنسبة للمهندسين ، يوفر Spark طريقة بسيطة لموازنة هذه التطبيقات عبر المجموعات ، ويخفي تعقيد برمجة الأنظمة الموزعة ، واتصالات الشبكة ، والتسامح مع الأخطاء. يمنحهم النظام تحكمًا كافيًا لمراقبة التطبيقات وفحصها وضبطها مع السماح لهم بتنفيذ المهام الشائعة بسرعة. تجعل الطبيعة المعيارية لواجهة برمجة التطبيقات (استنادًا إلى تمرير مجموعات الكائنات الموزعة) من السهل تحليل العمل في مكتبات قابلة لإعادة الاستخدام واختبارها محليًا.

يختار مستخدمو Spark استخدامه لتطبيقات معالجة البيانات الخاصة بهم لأنه يوفر مجموعة متنوعة من الوظائف ، ويسهل التعلم والاستخدام ، وهو ناضج وموثوق.

# تاريخ موجز Spark

Spark هو مشروع مفتوح المصدر تم بناؤه وصيانته من قبل مجتمع مزدهر ومتنوع من المطورين. إذا كنت أنت أو مؤسستك تحاول استخدام Spark لأول مرة ، فقد تكون مهتمًا بتاريخ المشروع. بدأ Spark في عام 2009 كمشروع بحثي في UC Berkeley RAD Lab ، ليصبح لاحقًا AMPLab. كان الباحثون في المختبر يعملون سابقًا على Hadoop Map‐ Reduce ، ولاحظوا أن MapReduce كان غير فعال لوظائف الحوسبة التكرارية والتفاعلية. وهكذا ، من البداية ، تم تصميم Spark ليكون سريعًا للاستعلامات التفاعلية والخوارزميات التكرارية ، مما يوفر أفكارًا مثل دعم التخزين في الذاكرة واسترداد الأخطاء بشكل فعال.

تم نشر الأوراق البحثية حول Spark في المؤتمرات الأكاديمية وبعد وقت قصير من إنشائها في عام 2009 ، كانت بالفعل أسرع بـ 10-20 مرة من MapReduce في وظائف معينة.

كان بعض مستخدمي Spark الأوائل مجموعات أخرى داخل جامعة كاليفورنيا في بيركلي ، بما في ذلك باحثو التعلم الآلي مثل مشروع Mobile Millennium ، الذي استخدم Spark لمراقبة الازدحام المروري والتنبؤ به في منطقة خليج سان فرانسيسكو. ومع ذلك ، في وقت قصير جدًا ، بدأت العديد من المنظمات الخارجية في استخدام Spark ، واليوم ، تُدرج أكثر من 50 منظمة نفسها على صفحة Spark PoweredBy ، ويتحدث العشرات عن حالات استخدامها في أحداث مجتمع Spark مثل Spark Meetups و Spark Summit. بالإضافة إلى جامعة كاليفورنيا في بيركلي ، فإن المساهمين الرئيسيين في Spark هم Databricks و Yahoo! و Intel.

في عام 2011 ، بدأ AMPLab في تطوير مكونات عالية المستوى على Spark ، مثل Shark (Hive on Spark) 1 و Spark Streaming. يشار إلى هذه المكونات وغيرها في بعض الأحيان باسم Berkeley Data Analytics Stack (BDAS).

تم فتح Spark لأول مرة في مارس 2010 ، وتم نقله إلى Apache Soft‐ Ware Foundation في يونيو 2013 ، حيث أصبح الآن مشروعًا عالي المستوى.

# خصائص Spark

تتمحور فلسفة سبارك في التصميم حول أربع خصائص رئيسية:

## سرعة

## سهولة الاستعمال

## نمطية

## القابلية للتوسعة

## السرعة

لقد سعى سبارك إلى تحقيق هدف السرعة بعدة طرق. أولاً ، يستفيد تنفيذه الداخلي بشكل كبير من الخطوات الضخمة الأخيرة لصناعة الأجهزة في تحسين سعر وأداء وحدات المعالجة المركزية والذاكرة. أصبحت خوادم السلع اليوم رخيصة ، مع مئات الجيجابايت من الذاكرة ، وعدة مراكز ، ونظام التشغيل الأساسي القائم على نظام التشغيل Unix الذي يستفيد من كفاءة تعدد مؤشرات الترابط والمعالجة المتوازية. تم تحسين إطار العمل للاستفادة من كل هذه العوامل.

ثانيًا ، تبني Spark حسابات الاستعلام الخاصة بها كرسم بياني دوري موجَّه (DAG) ؛ يقوم برنامج جدولة DAG ومحسن الاستعلام بإنشاء رسم بياني حسابي فعال يمكن عادةً تحليله إلى مهام يتم تنفيذها بشكل متوازٍ عبر العاملين في المجموعة.

مع الاحتفاظ بجميع النتائج الوسيطة في الذاكرة ومحدودية إدخال / إخراج القرص الخاص بها ، فإن هذا يمنحها دفعة هائلة في الأداء.

## سهولة الاستعمال

يحقق Spark البساطة من خلال توفير تجريد أساسي لهيكل بيانات منطقي بسيط يسمى مجموعة البيانات الموزعة المرنة (RDD) والتي يتم بناء جميع تجريدات البيانات المنظمة عالية المستوى الأخرى ، مثل DataFrames ومجموعات البيانات. من خلال توفير مجموعة من التحويلات والإجراءات كعمليات ، تقدم Spark نموذج برمجة بسيطًا يمكنك استخدامه لبناء تطبيقات البيانات الضخمة باللغات العائلية.

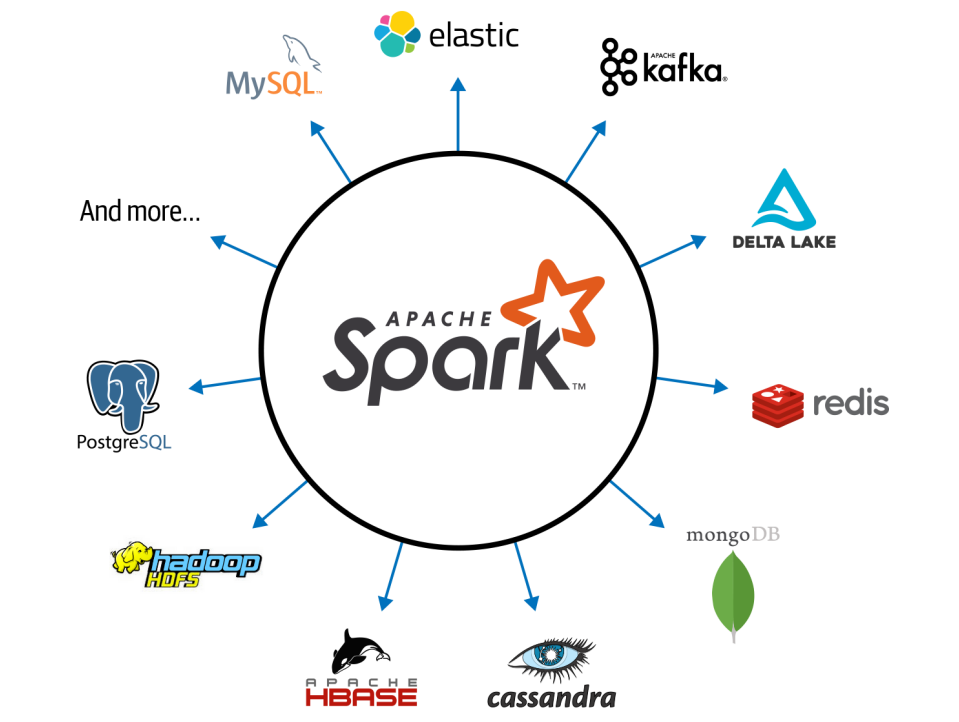
## نمطية

يمكن تطبيق عمليات Spark عبر العديد من أنواع أحمال العمل والتعبير عنها في أي من لغات البرمجة المدعومة: تقدم Scala و Java و Python و SQL و R. Spark مكتبات موحدة مع واجهات برمجة تطبيقات موثقة جيدًا تتضمن الوحدات النمطية التالية كمكونات أساسية : Spark SQL و Spark Structured Streaming و Spark MLlib و GraphX ، التي تجمع بين جميع أحمال العمل التي تعمل تحت محرك واحد. سنلقي نظرة فاحصة على كل هذه في القسم التالي.

يمكنك كتابة تطبيق Spark واحد يمكنه القيام بكل شيء - لا حاجة لمحركات مميزة لأحمال عمل متباينة ، ولا حاجة لتعلم واجهات برمجة تطبيقات منفصلة. مع Spark ، تحصل على محرك معالجة موحد لأحمال عملك.

## قابلية التوسع

يركز Spark على محركه الحسابي السريع والمتوازي بدلاً من التخزين. على عكس Apache Hadoop ، الذي يتضمن كلاً من التخزين والحساب ، يفصل Spark الاثنين. هذا يعني أنه يمكنك استخدام Spark لقراءة البيانات المخزنة في عدد لا يحصى من المصادر - Apache Hadoop و Apache Cassandra و Apache HBase و MongoDB و Apache Hive و RDBMSs والمزيد - ومعالجتها كلها في الذاكرة. يمكن أيضًا توسيع برامج Spark's DataFrameReaders و DataFrame Writers لقراءة البيانات من مصادر أخرى ، مثل Apache Kafka و Kinesis و Azure Storage و Amazon S3 ، إلى تجريدها المنطقي للبيانات ، والذي يمكن أن تعمل عليه.

يحتفظ مجتمع مطوري Spark بقائمة من حزم Spark التابعة لجهات خارجية كجزء من النظام البيئي المتنامي (انظر الشكل 2). يشتمل هذا النظام البيئي الغني للحزم على موصلات Spark لمجموعة متنوعة من مصادر البيانات الخارجية ، ومراقبة الأداء ، والمزيد.

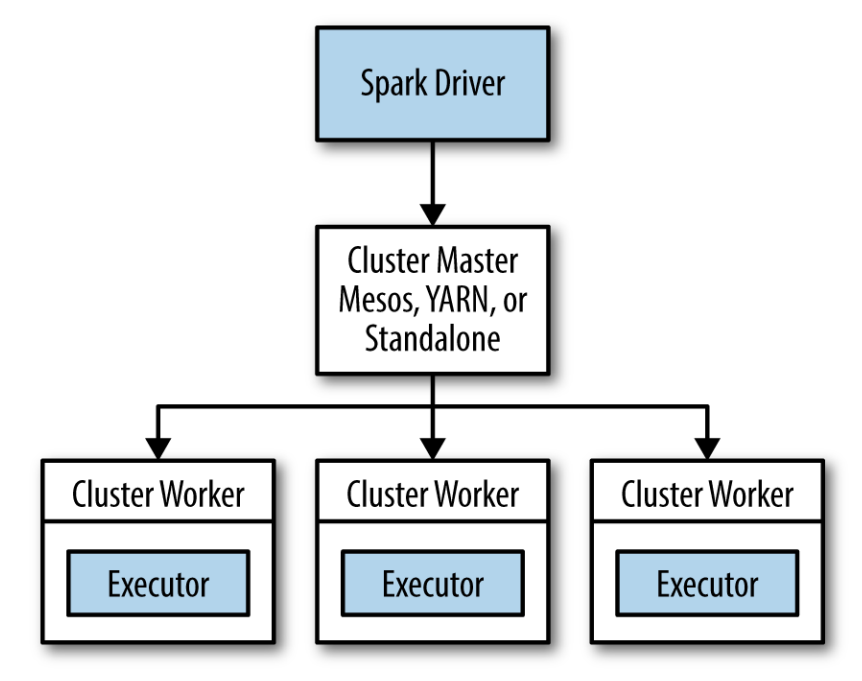
شكل 2 - النظام البيئي للموصلات Apache Spark

الفصل الثاني

**Spark Architecture**

# ماهي هيكلية Spark؟

هيكلية Spark في الوضع الموزع (كما هو موضح في الشكل 3).

في الوضع الموزع ، يستخدم Spark بنية رئيسية / تابعة مع منسق مركزي واحد والعديد من العمال الموزعين. المنسق المركزي يسمى المشغل. يتواصل المشغل مع عدد كبير محتمل من العمال الموزعين الذين يطلق عليهم المنفذون. يتم تشغيل برنامج التشغيل في عملية Java الخاصة به وكل منفذ هو عملية Java منفصلة. يُطلق على المشغل ومنفذه معًا تطبيق Spark.

شكل 3 - مكونات تطبيق Spark الموزع

يتم تشغيل تطبيق Spark على مجموعة من الأجهزة باستخدام خدمة خارجية تسمى مدير الكتلة. كما لوحظ ، يتم حزم Spark مع مدير كتلة مضمن يسمى مدير الكتلة المستقل. يعمل Spark أيضًا مع Hadoop YARN و Apache Mesos ، وهما مديرا مجموعات مفتوحة المصدر شهيران.

## المشغل The Driver

المشغل هو العملية التي يتم فيها تشغيل main() الرئيسية لبرنامجك. إن العملية التي تقوم بتشغيل كود المستخدم هي التي تقوم بإنشاء SparkContext ، وإنشاء RDDs ، والتحويلات والإجراءات لكل النماذج. عند تشغيل Spark shell ، تكون قد أنشأت برنامج تشغيل (إذا كنت تتذكر ، فإن Spark shell تأتي محملة مسبقًا بنص SparkCon‐ يسمى sc). بمجرد انتهاء برنامج التشغيل ، يتم الانتهاء من التطبيق.

عندما يعمل المشغل، فإنه يؤدي مهمتين:

### تحويل برنامج المستخدم إلى مهام

مشغل Spark مسؤول عن تحويل برنامج المستخدم إلى وحدات للتنفيذ الفعلي تسمى المهام. على مستوى عالٍ ، تتبع جميع برامج Spark نفس البنية: فهي تنشئ RDDs من بعض المدخلات ، وتستمد RDDs جديدة من تلك التي تستخدم التحويلات ، وتنفذ إجراءات لجمع البيانات أو حفظها. ينشئ البرنامج الوامض ضمنيًا رسمًا بيانيًا دائريًا منطقيًا (DAG) للعمليات. عندما يعمل المشغل ، فإنه يحول هذا الرسم البياني المنطقي إلى خطة تنفيذ فعلية.

يقوم Spark بإجراء العديد من التحسينات ، مثل تحويلات خريطة "التسلسل" معًا لدمجها ، وتحويل الرسم البياني للتنفيذ إلى مجموعة من المراحل. كل مرحلة ، بدورها ، تتكون من مهام متعددة. يتم تجميع المهام وتقليصها مسبقًا لإرسالها إلى المجموعة. المهام هي أصغر وحدة عمل في Spark ؛ يمكن لبرنامج المستخدم العادي إطلاق مئات أو آلاف المهام الفردية.

### جدولة المهام على المنفذين

بالنظر إلى خطة التنفيذ المادية ، يجب على مشغل Spark تنسيق جدولة المهام الفردية على المنفذين. عندما يبدأ المنفذون بتسجيل أنفسهم مع المشغل ، بحيث يكون لديه رؤية كاملة لمنفذي التطبيق في جميع الأوقات. يمثل كل منفذ عملية قادرة على تشغيل المهام وتخزين بيانات RDD.

سينظر برنامج تشغيل Spark إلى المجموعة الحالية من المنفذين ويحاول جدولة كل مهمة في مكان مناسب ، بناءً على وضع البيانات. عند تنفيذ المهام ، قد يكون لها تأثير جانبي لتخزين البيانات المخزنة مؤقتًا. يتتبع برنامج التشغيل أيضًا موقع البيانات المخزنة مؤقتًا ويستخدمها لجدولة المهام المستقبلية التي تصل إلى تلك البيانات.

يعرض برنامج التشغيل معلومات حول تطبيق Spark قيد التشغيل من خلال واجهة ويب ، والتي تتوفر افتراضيًا في المنفذ 4040. على سبيل المثال ، في الوضع المحلي ، تتوفر واجهة المستخدم هذه على http: // localhost: 4040.

## المنفذ Executors

منفذي شرارة هم عمليات عاملة مسؤولة عن تشغيل المهام الفردية في وظيفة سبارك معينة. يتم إطلاق المنفذين مرة واحدة في بداية تطبيق Spark ويتم تشغيلهم عادةً طوال عمر التطبيق بالكامل ، على الرغم من أن تطبيقات Spark يمكن أن تستمر في حالة فشل المنفذين. المنفذون بدورين. أولاً ، يقومون بتشغيل المهام التي يتكون منها التطبيق وإرجاع النتائج إلى المشغل. ثانيًا ، أنها توفر تخزينًا في الذاكرة لـ RDDs التي يتم تخزينها مؤقتًا بواسطة برامج المستخدم ، من خلال خدمة تسمى Block Manager تعيش داخل كل منفذ. نظرًا لأنه يتم تخزين RDDs مؤقتًا مباشرة داخل المنفذين ، يمكن تشغيل المهام جنبًا إلى جنب مع البيانات المخزنة مؤقتًا.

# إدارة المجاميع Cluster Manager

يعتمد Spark على مدير الكتلة لإطلاق المنفذين ، وفي بعض الحالات ، لتشغيل المشغل. مدير الكتلة هو مكون قابل للتوصيل في Spark. يسمح ذلك لـ Spark بالعمل على رأس مديرين خارجيين مختلفين ، مثل YARN و Mesos ، بالإضافة إلى مدير مجموعة مستقل مدمج.

# تشغيل البرنامج Launching a Program

بغض النظر عن مدير المجموعة الذي تستخدمه ، يوفر Spark برنامجًا نصيًا واحدًا يمكنك استخدامه لإرسال برنامجك إليه يسمى إرسال شرارة. من خلال خيارات متنوعة ، يمكن لـ Sparksubmit الاتصال بمديري مجموعة مختلفين والتحكم في عدد الموارد التي يحصل عليها تطبيقك. بالنسبة لبعض مديري الكتلة ، يمكن لـ spark-submit تشغيل برنامج التشغيل داخل المجموعة (على سبيل المثال ، على عقدة عاملة YARN) ، بينما بالنسبة للآخرين ، يمكن تشغيلها فقط على جهازك المحلي.

# الملخص

لتلخيص المفاهيم الواردة في هذا القسم ، دعنا نتصفح الخطوات الدقيقة التي تحدث عند تشغيل تطبيق Spark على مجموعة:

1. يقدم المستخدم طلبًا باستخدام spark-submit.
2. spark-submit يقوم بتشغيل برنامج التشغيل ويستدعي الطريقة main() المحددة من قبل المستخدم.
3. يقوم برنامج التشغيل بالاتصال بمدير الكتلة لطلب الموارد لبدء تشغيل المنفذين.
4. يقوم مدير الكتلة بتشغيل المنفذين نيابة عن برنامج المشغل.
5. يتم تشغيل عملية المشغل من خلال تطبيق المستخدم. بناءً على إجراءات RDD والتحولات في البرنامج ، يرسل المشغل العمل إلى المنفذين في شكل مهام.
6. يتم تشغيل المهام على عمليات المنفذ لحساب النتائج وحفظها.
7. إذا تم إنهاء طريقة () main لبرنامج التشغيل أو كانت تستدعي SparkContext.stop ، فسوف تقوم بإنهاء المنفذين وإصدار الموارد من مدير المجموعة.

# Cluster Managers

يمكن لـ Spark تشغيل مجموعة متنوعة من مديري المجموعات للوصول إلى الأجهزة الموجودة في المجموعة. إذا كنت ترغب فقط في تشغيل Spark بمفرده على مجموعة من الأجهزة ، فإن وضع Standalone المدمج هو أسهل طريقة لنشره. ومع ذلك ، إذا كان لديك مجموعة ترغب في مشاركتها مع التطبيقات الموزعة الأخرى (على سبيل المثال ، وظائف Spark و Hadoop MapRe duce) ، فيمكن أيضًا تشغيل Spark على مديري مجموعة شهيرين: Hadoop YARN و Apache Mesos. أخيرًا ، للنشر على Amazon EC2 ، يأتي Spark مزودًا بنصوص برمجية مدمجة تطلق مجموعة مستقلة وخدمات دعم متنوعة. في هذا القسم ، سنغطي كيفية تشغيل Spark في كل من هذه البيئات.

# Standalone Cluster Manager

يوفر مدير Spark's Standalone طريقة بسيطة لتشغيل التطبيقات على مجموعة. وهو يتألف من سيد وعمال متعددين ، ولكل منهم مقدار مهيأ من الذاكرة ونوى وحدة المعالجة المركزية. عند تقديم طلب ، يمكنك اختيار مقدار الذاكرة التي سيستخدمها منفذوها ، بالإضافة إلى العدد الإجمالي للنواة عبر جميع المنفذين.

# Hadoop YARN

YARN هو مدير مجموعة تم تقديمه في Hadoop 2.0 والذي يسمح لأطر عمل معالجة بيانات متنوعة بالعمل على مجموعة موارد مشتركة ، ويتم تثبيته عادةً على نفس العقد مثل نظام ملفات Hadoop (HDFS). يعد تشغيل Spark على YARN في هذه البيئات مفيدًا لأنه يتيح لـ Spark الوصول إلى بيانات HDFS بسرعة ، على نفس العقد حيث يتم تخزين البيانات.

يعد استخدام YARN في Spark أمرًا بسيطًا: يمكنك تعيين متغير بيئة يشير إلى دليل تكوين Hadoop الخاص بك ، ثم إرسال المهام إلى عنوان URL رئيسي خاص باستخدام spark-submit.

# Apache Mesos

Apache Mesos هو مدير مجموعة للأغراض العامة يمكنه تشغيل كل من أحمال عمل التحليلات والخدمات طويلة المدى (مثل تطبيقات الويب أو مخازن المفاتيح / القيمة) على مجموعة.

على عكس مديري المجموعات الآخرين ، يقدم Mesos وضعين لمشاركة الموارد بين المنفذين في نفس المجموعة. في وضع "الحبيبات الدقيقة" ، وهو الوضع الافتراضي ، يقوم المنفذون بتوسيع وتقليص عدد وحدات المعالجة المركزية التي يطالبون بها من Mesos أثناء قيامهم بتنفيذ مهام لطيفة ، وبالتالي يمكن للجهاز الذي يقوم بتشغيل العديد من المنفذين مشاركة موارد وحدة المعالجة المركزية بينهم بشكل ديناميكي. في الوضع "coarse-grained" ، يخصص Spark عددًا ثابتًا من وحدات المعالجة المركزية لكل منفذ مقدمًا ولا يطلقها أبدًا حتى ينتهي التطبيق ، حتى إذا كان المنفذ لا يقوم حاليًا بتشغيل المهام. يمكنك تمكين وضع الحبيبات الخشنة عن طريق تمرير --conf spark.mesos.coarse = true to spark-submit.

يكون وضع Mesos الدقيق جذابًا عندما يشترك العديد من المستخدمين في مجموعة لتشغيل أحمال عمل تفاعلية مثل الصدفة ، لأن التطبيقات ستقلل من عدد النوى الخاصة بها عندما لا تقوم بعمل وتسمح لبرامج المستخدمين الآخرين باستخدام العنقودية. ومع ذلك ، فإن الجانب السلبي هو أن جدولة المهام من خلال الوضع الدقيق يضيف مزيدًا من وقت الاستجابة (لذلك قد تتأثر التطبيقات ذات زمن الوصول المنخفض جدًا مثل Spark Streaming) ، وقد تحتاج التطبيقات إلى الانتظار بعض الوقت حتى تصبح وحدة المعالجة المركزية مجانية " up "مرة أخرى عندما يكتب المستخدم أمرًا جديدًا. لاحظ ، كيف من أي وقت مضى ، أنه يمكنك استخدام مزيج من أوضاع الجدولة في نفس مجموعة Mesos (على سبيل المثال ، قد يكون لبعض تطبيقات Spark الخاصة بك مجموعة spark.mesos.coarse على صحيح والبعض الآخر قد لا).

# Amazon EC2

يأتي Spark مع برنامج نصي مدمج لإطلاق المجموعات على Amazon EC2. يقوم هذا البرنامج النصي بتشغيل مجموعة من العقد ثم يقوم بتثبيت مدير الكتلة المستقل عليها ، لذلك بمجرد إنشاء الكتلة ، يمكنك استخدامها. بالإضافة إلى ذلك ، يقوم البرنامج النصي EC2 بإعداد خدمات الدعم مثل HDFS و Tachyon و Ganglia لمراقبة مجموعتك.

# سبارك: ماذا يوجد تحت RDD؟

RDD هو التجريد الأساسي في Spark. هناك ثلاث خصائص حيوية مرتبطة بـ RDD:

• التبعيات

• أقسام (مع بعض المعلومات المحلية)

• وظيفة الحساب: القسم => التكرار [T]

الثلاثة كلها جزء لا يتجزأ من نموذج API لبرمجة RDD البسيط الذي يتم بناء جميع وظائف المستوى الأعلى بناءً عليه. أولاً ، مطلوب قائمة من التبعيات التي ترشد Spark حول كيفية إنشاء RDD مع مدخلاته. عند الضرورة لإعادة إنتاج النتائج ، يمكن لـ Spark إعادة إنشاء RDD من هذه التبعيات وتكرار العمليات عليه. تعطي هذه الخاصية مرونة RDDs.

ثانيًا ، توفر الأقسام لـ Spark القدرة على تقسيم العمل لموازنة الحوسبة على الأقسام عبر المنفذين. في بعض الحالات - على سبيل المثال ، القراءة من HDFS - ستستخدم Spark معلومات المنطقة لإرسال العمل إلى منفذين قريبين من البيانات. بهذه الطريقة يتم نقل بيانات أقل عبر الشبكة.

وأخيرًا ، يحتوي RDD على وظيفة حسابية تنتج متكررًا [T] للبيانات التي سيتم تخزينها في RDD.

بسيطة وأنيقة! ومع ذلك ، هناك مشكلتان في هذا النموذج الأصلي. لأحدهما ، فإن وظيفة الحساب (أو الحساب) معتم بالنسبة لـ Spark. أي أن Spark لا تعرف ما تفعله في وظيفة الحساب. سواء كنت تقوم بتنفيذ صلة أو تصفية أو تحديد أو تجميع ، فإن Spark يراها فقط كتعبير lambda. مشكلة أخرى هي أن نوع البيانات Iterator [T] معتم أيضًا لـ Python RDDs ؛ يعرف Spark فقط أنه كائن عام في Python.

علاوة على ذلك ، نظرًا لأنه غير قادر على فحص الحساب أو التعبير في الوظيفة ، فإن Spark ليس لديه طريقة لتحسين التعبير - ليس لديه فهم لهدفه. وأخيرًا ، ليس لدى Spark معرفة بنوع البيانات المحدد في T. To Spark ، فهو كائن معتم ؛ ليس لديه فكرة عما إذا كنت تقوم بالوصول إلى عمود من نوع معين داخل كائن. لذلك ، كل ما يمكن لـ Spark القيام به هو إجراء تسلسل للكائن المعتم كسلسلة من البايت ، دون استخدام أي تقنيات لضغط البيانات.

# هيكل سبارك Structuring Spark

قدم Spark 2.x بعض المخططات الرئيسية لهيكلة Spark. الأول هو التعبير عن الجمل باستخدام الأنماط الشائعة الموجودة في تحليل البيانات. يتم التعبير عن هذه الأنماط كعمليات عالية المستوى مثل التصفية والاختيار والعد والتجميع والتوسيط والتجميع. هذا يوفر المزيد من الوضوح والبساطة.

تم تضييق هذه الخصوصية بشكل أكبر من خلال استخدام مجموعة من المشغلين المشتركين في DSL. من خلال مجموعة من العمليات في DSL ، والمتاحة كواجهات برمجة تطبيقات في أدلة Spark اللغوية المدعومة (Java و Python و Spark و R و SQL) ، تتيح لك هذه المشغلات إخبار Spark بما ترغب في حسابه ببياناتك ، ونتيجة لذلك ، يمكنه إنشاء خطة استعلام فعالة للتنفيذ.

والمخطط النهائي للترتيب والهيكل هو السماح لك بترتيب بياناتك في تنسيق جدولي ، مثل جدول أو جدول بيانات SQL.

# متى يستخدم RDD - When to Use RDDs

سيستمر دعم واجهة برمجة تطبيقات RDD ، على الرغم من أن جميع أعمال التطوير المستقبلية في Spark 2.x و Spark 3.0 ستستمر في الحصول على واجهة DataFrame ودلالات بدلاً من استخدام RDDs.

هناك بعض السيناريوهات التي قد ترغب في التفكير فيها باستخدام RDD ، مثل عندما:

• تستخدم حزمة جهة خارجية تمت كتابتها باستخدام RDDs

• يمكن أن تتخلى عن تحسين الكود ، والاستخدام الفعال للمساحة ، ومزايا الأداء المتوفرة مع DataFrames ومجموعات البيانات

• تريد توجيه تعليمات Spark بدقة حول كيفية إجراء استعلام

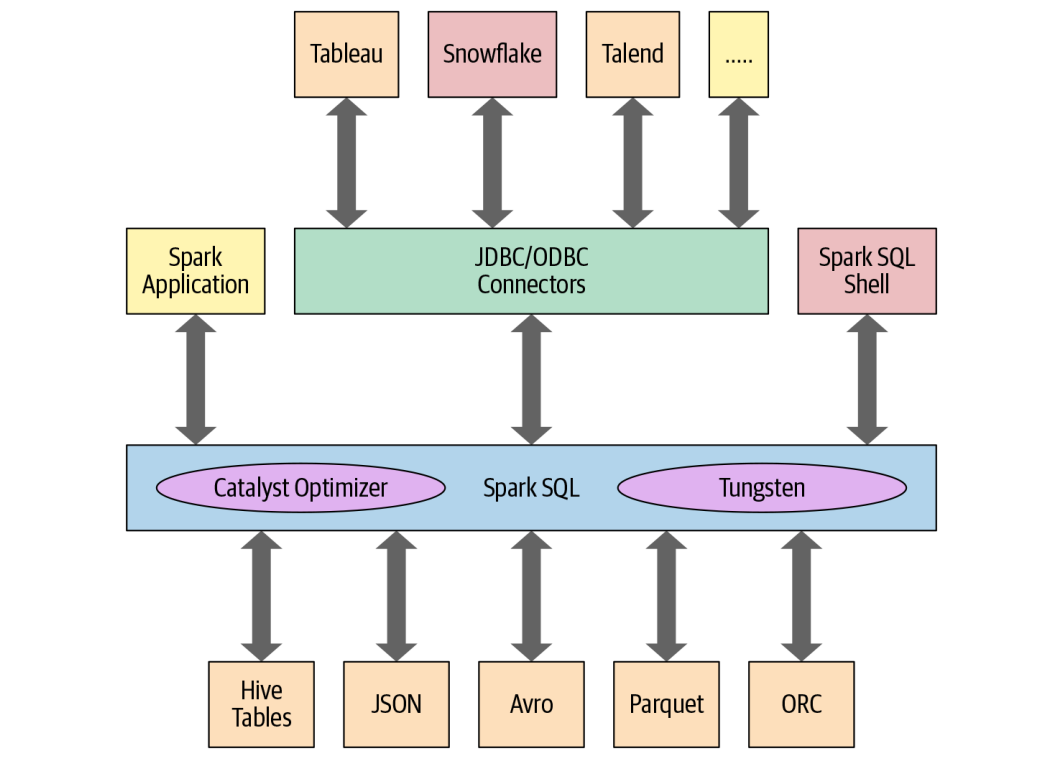
علاوة على ذلك ، يمكنك التنقل بسلاسة بين DataFrames أو Datasets و RDDs كما تشاء باستخدام استدعاء طريقة API بسيط ، df.rdd. (لاحظ ، مع ذلك ، أن هذا له تكلفة ويجب تجنبه ما لم يكن ذلك ضروريًا.) بعد كل شيء ، يتم إنشاء إطارات البيانات ومجموعات البيانات فوق RDDs ، وتتحلل إلى رمز RDD المضغوط أثناء إنشاء رمز المرحلة الكاملة.

# Spark SQL والمحرك الأساسي

على المستوى البرمجي ، يسمح Spark SQL للمطورين بإصدار ANSI SQL: 2003 - استعلامات قابلة للتطبيق على البيانات المهيكلة باستخدام مخطط. منذ تقديمها في Spark 1.3 ، تطورت Spark SQL إلى محرك أساسي تم بناء العديد من الوظائف الهيكلية عالية المستوى عليه. بصرف النظر عن السماح لك بإصدار استعلامات تشبه SQL على بياناتك ، فإن محرك Spark SQL:

* يوحد مكونات Spark ويسمح بالتجريد إلى DataFrames / Datasets في Java و Scala و Python و R ، مما يبسط العمل مع مجموعات البيانات المنظمة.
* يتصل بمخزن Apache Hive والجداول.
* يقرأ ويكتب البيانات المهيكلة بمخطط محدد من تنسيقات الملفات المهيكلة (JSON ، CSV ، Text ، Avro ، Parquet ، ORC ، إلخ) ويحول البيانات إلى جداول مؤقتة.
* يقدم قذيفة Spark SQL تفاعلية لاستكشاف البيانات بسرعة.
* يوفر جسرًا إلى (ومن) الأدوات الخارجية عبر موصلات JDBC / ODBC القياسية لقاعدة البيانات.
* يولد خطط استعلام محسنة ورمز مضغوط لـ JVM ، للتنفيذ النهائي.

يوضح الشكل 4 المكونات التي يتفاعل معها Spark SQL لتحقيق كل هذا.

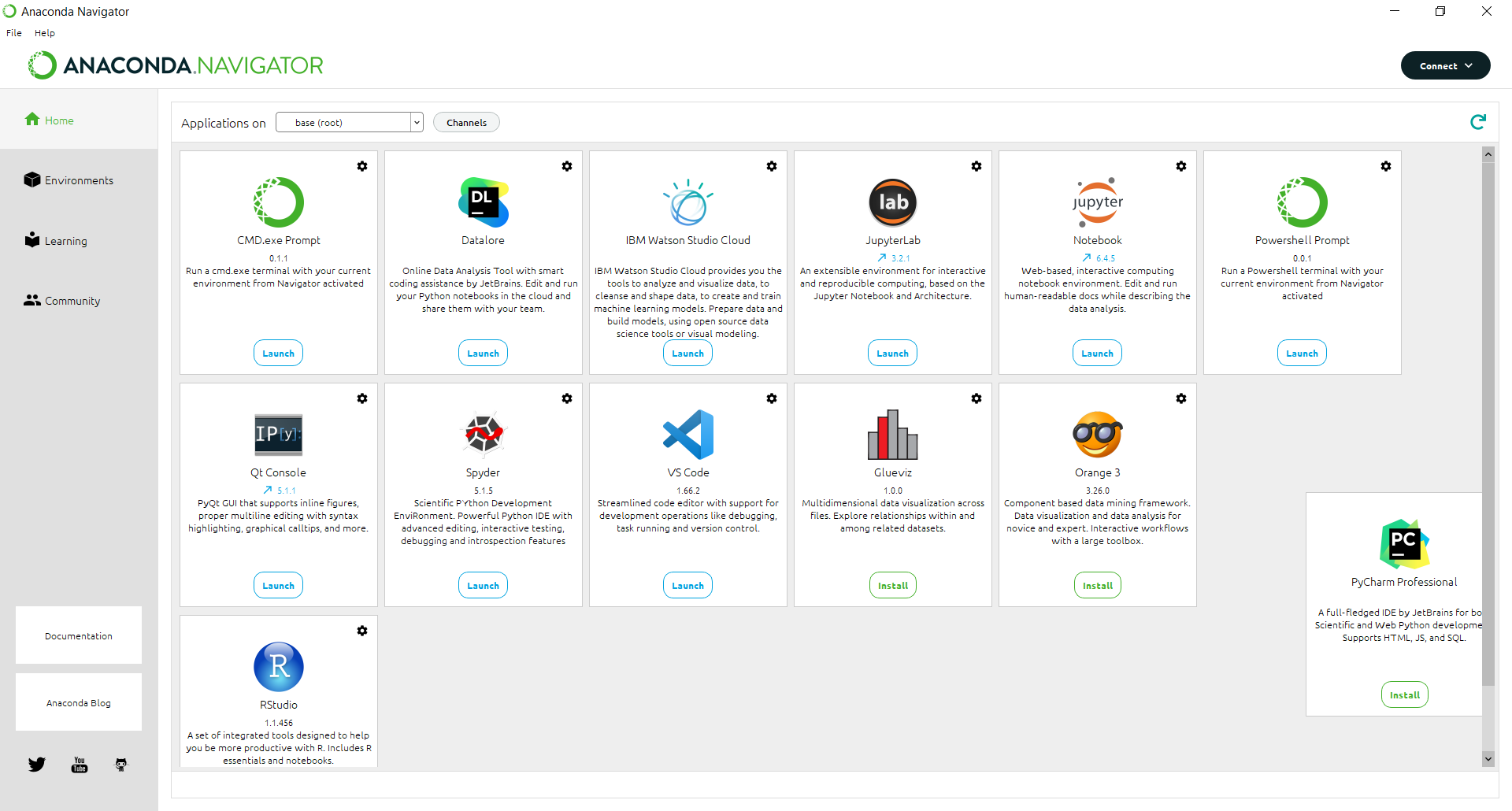
شكل 4 - Spark SQL ومكدسها

في قلب محرك Spark SQL يوجد محسن Catalyst و Project Tungsten. معًا ، تدعم هذه الملفات عالية المستوى DataFrame و Dataset APIs واستعلامات SQL.

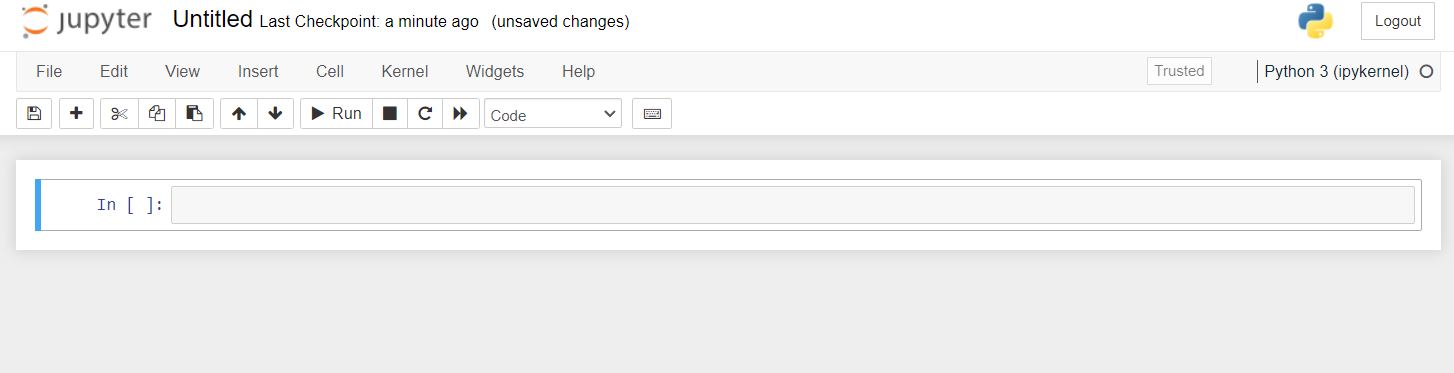
الفصل الثالث

**تطبيقات في Spark**

# تطبيق Anaconda

في هذا المشروع قمنا بعمل تثبيت تطبيق Anaconda على الويندوز لتشغيل سيرفر Spark عليه على المتصفح كما موضح في الشكل 5. حيث تم اعتماد لغة Python معه. لذلك تم العمل على Jupyter Notebook لكتابة الشفرات البرمجية الخاصة بـ Spark ولغة Python.

شكل 5 – واجهة تطبيق Anaconda

حيث يتم تشغيله على المتصفح ليعرض الملفات والمجلدات في الحاسبة ليتم انشاء ملفات جديدة بلغة Python لكتابة الكودات البرمجية في بيئة Spark بصيغة (.ipynb) كما في الشكل 6. هذا النوع من الملفات يكون خاص بلغة Python بأستخدام Jupyter Notebook.

شكل 6 – واجهة Jupyter Notebook

# اساسيات RDD

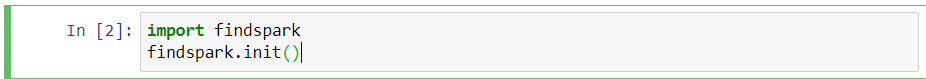
وهو مجموعة من أجزاء الشفرات والكائنات التوزيعية. وكل RDD مقسم الى مجموعة أجزاء، والتي من الممكن ان تحسب عقد مختلفة من المجاميع او الكتل. وكل RDD يستطيع ان يحتوي على نوع معين من لغات البرمجة Python, Java, Scala او غيرها من اللغات التي يعرفها المستخدم.

# امثلة تطبيقية

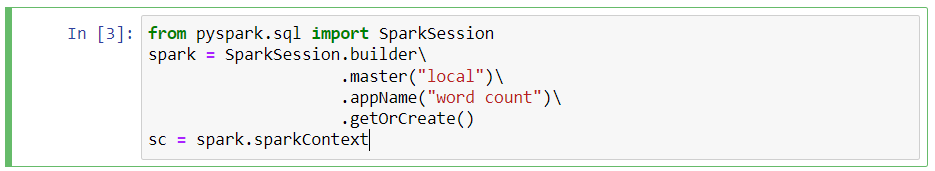
سنتناول تطبيقيين تم عملهم باستخدام Spark بالتعامل مع لغة Python لبيان الية العمل على Spark.

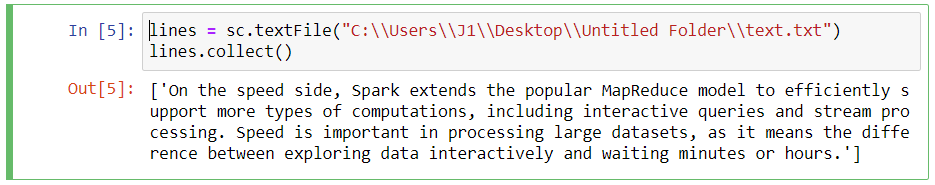
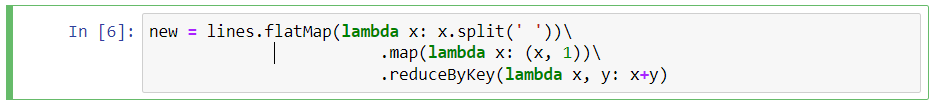
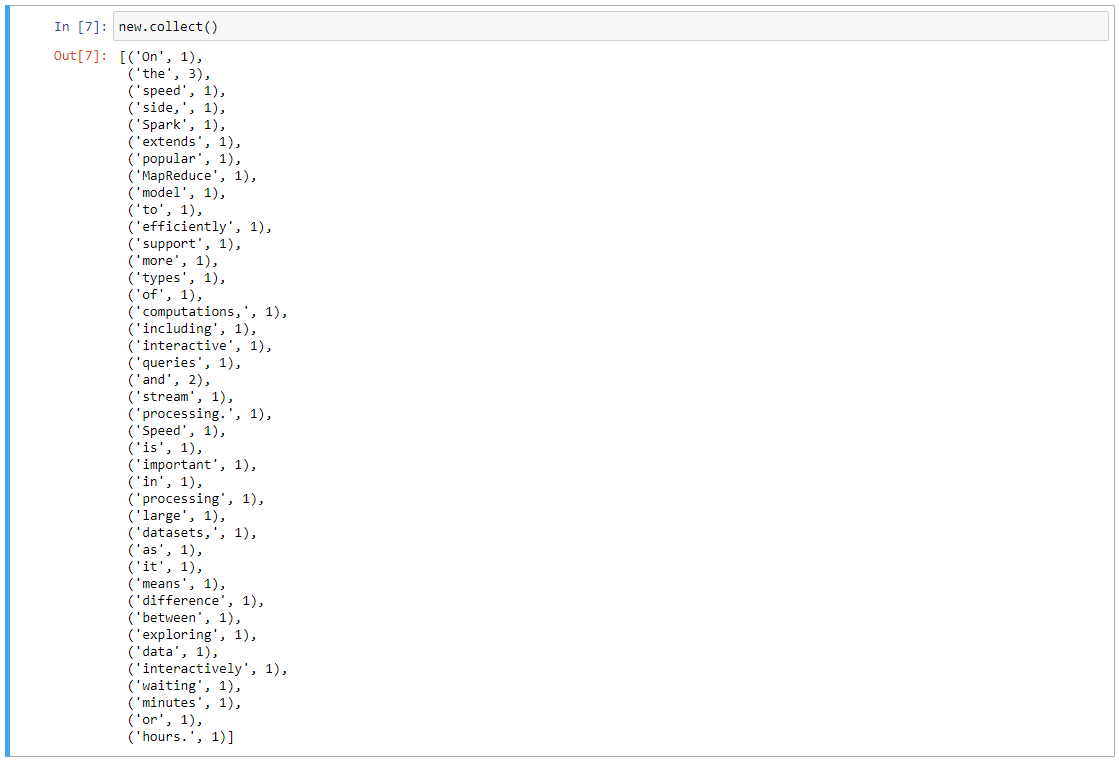
## تطبيق حساب تكرار الكلمات في الجملة

تم عمل التطبيق باستخدام بيئة Spark بلغة Python والذي سيتم من خلاله تحديد ملف نصي معين مخزن مسبقا في الحاسب. حيث سيقوم بعمل كتل من الكلمات من خلال النص الموجود داخل الملف النصي. حيث سيحتوي على كل كلمة وعدد تكراراتها داخل النص في الملف. مثل هكذا تطبيقات مهمة جدا في الكثير من خوارزميات التنقيب على البيانات Data Mining او Big Data او تعليم الالة Machine Learning مثل خوارزمية Clustering او Classification وغيرها من الخوارزميات. حيث يتم استخدام بيئة Spark وذلك لسرعة الأداء فيه.

**الكود البرمجي**

1. يتم استدعاء حزمة **findspark** لتهيئة بيئة Spark.

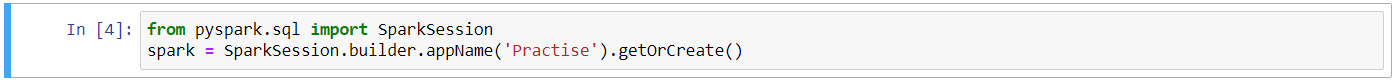


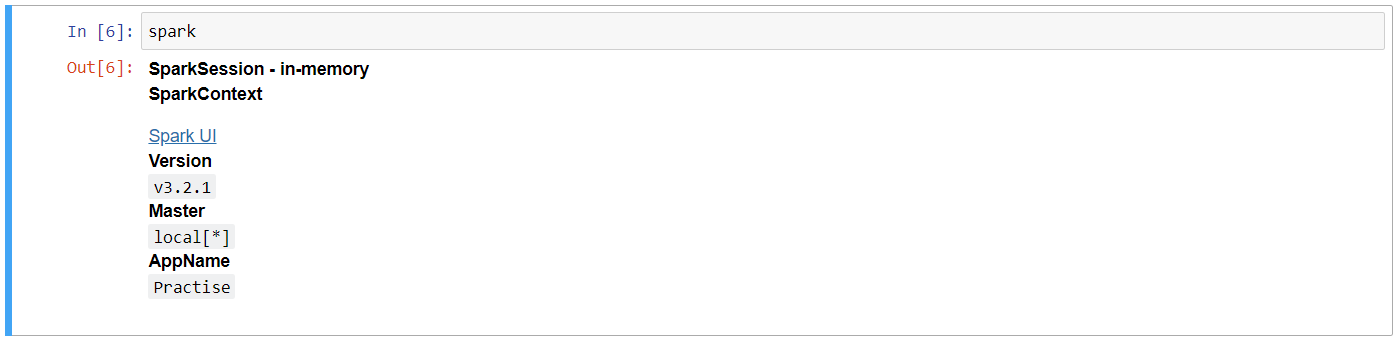
1. يتم استدعاء حزمة **SparkSession** لعمل جلسة جديدة بنافذة محلية local وتسمية التطبيق باسم معين وانشائه. ثم عمل كائن جديد منه باسم **sc**. وذلك للاستدعاء بكتابة الشفرات البرمجية الخاصة بـ Spark بالاعتماد على لغة البرمجة Python.
2. في هذا الجزء من العقدة اعله يتم تحميل ملف من الحاسبة. حيث يتم استخدام الدالة **textFile** لتحميل الملف من الحاسبة بالمسار المحدد. حيث يتم استخدام \\ وذلك لان \ يعتبر رمز خروج، ففي هذه الحالة يوقف عمله ليعتبر رمز فاصل بين المجلدات. بعد تحديد الملف يتم استخدام دالة **collect** ليتم عرض محتوى الملف.
3. في هذه العقدة يتم عمل دوارة map على النص الذي تم تحميله من الملف. ليتم تقطيع كل الكلمات بالاعتماد على الفراغ " " الذي يفصل بين كلمة وكلمة. ليتم اختبار كل كلمة فما اذا تكررت واحتساب لها التكرار في مصفوفة التكرارات للكلمات.
4. في هذه العقدة سيتم طباعة ناتج مصفوفة الكلمات المكررة الناتجة من النص المحدد.

## تطبيق أوامر SQL DataFrames

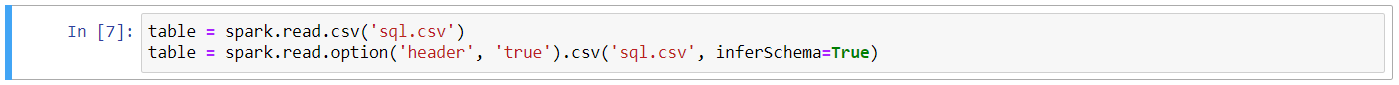
من التطبيقات المهمة باستخدام Spark هي التعامل بأوامر SQL على الجداول ليتم التعامل معها بشكل سهل.

**الكود البرمجي**

1. يتم استدعاء حزمة pyspark ليتم تنفيذ الكود الخاص بلغة Python في بيئة Spark. كذلك يتم pandas واختصارها بالكائن pd. تقوم هذه الحزمة بالتعامل مع ملفات الجداول.
2. نستخدم الدالة **read\_csv** ونحدد ملف الجدول. في هذا التطبيق قمنا بعمل جدول CSV وقمنا بملئه بالسجلات كم موضح أعلاه.
3. في هذه العقدة يتم عمل جلسة جديدة وتهيئته وانشائه.



1. هنا يتم عرض محتوى ومعلومات الجلسة التي قمنا بأنشائها.



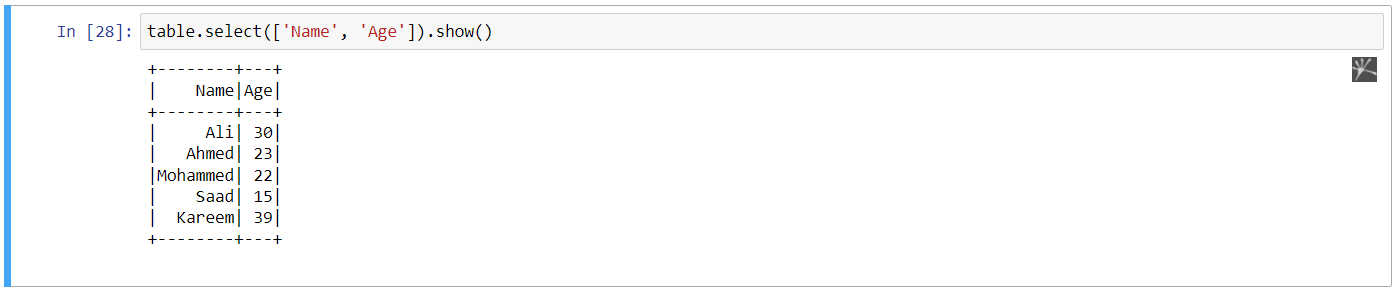
1. بعدها نقوم بقراءة نفس الملف لكن هذه المرة باستخدام بيئة Spark ليتم تنفيذ عليه أوامر DataFrames الخاصة بقواعد البيانات SQL. حيث تم عمل اعدادات لعرض الرأس للجدول. وكذلك التعامل مع النوع البياني الصحيح حسب البيانات الموجود في كل عمود.



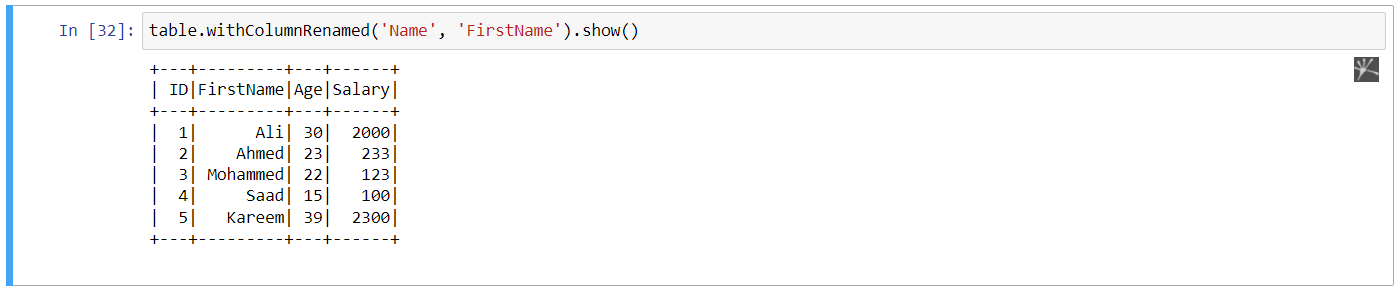
1. نقوم باستخدام دالة **printSchema** لعرض الاعمدة والانواع البيانية لكل عمود وبعض الخصائص الخاصة بها.

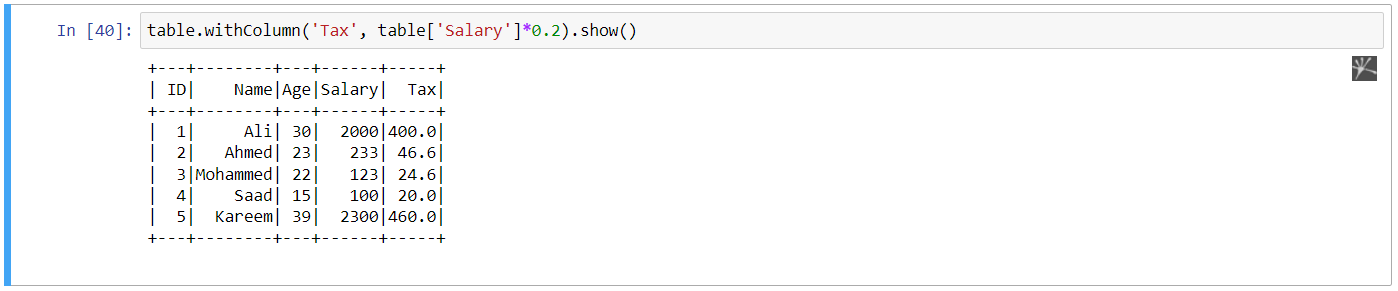


1. الدالة **show** تقوم بعرض البيانات الموجودة في الجدول.



1. باستخدام دالة **select** وهي مشابهة لأمر select في SQL لتحديد الاعمدة المراد عرضها او تحديد الشروط لعرضها.



1. الدالة withColumnRenamed تقوم بتغيير اسم العمود المحدد الى اسم جديد مع عرض الجدول.
2. يمكن إضافة عمود جديد باستخدام withColumn لتحديد اسمه والبيانات التي نريد اضافتها له.



1. يمكن حذف عمود معين باستخدام drop.



1. يمكن استخدام شروط لعرض الجدول حسب قيمة او شرط معين يتم اختياره باستخدام Filter.

الفصل الرابع

**الاستنتاجات**

# الاستنتاجات

العمل على تجزئة وتصفية البيانات في خوارزميات Big Data او Data Mining وحتى في تعليم الالة Machine Learning يحتاج وقت كبير لمعالجة البيانات. حيث ان العمل على هذه الخوارزميات في لغات البرمجة بشكل مباشر ممكن ان يكون بوقت بطيء. لكن تم ملاحظة سرعة أكبر بكثير بالعمل على الخوارزميات وحتى في نفس اللغات. لكن في بيئة Spark سيوفر وقت كثير ونتائج ومخرجات أسرع.

# المصادر

1. Lojkine, O. (2017). Оптимальные параметры для соединения данных на Apache Spark с фильтрами Блума. *ArXiv*.
2. Salloum, S., Dautov, R., Chen, X., Peng, P. X., & Huang, J. Z. (2016). Big data analytics on Apache Spark. *International Journal of Data Science and Analytics*, *1*(3–4), 145–164. https://doi.org/10.1007/s41060-016-0027-9
3. García-Gil, D., Ramírez-Gallego, S., García, S., & Herrera, F. (2017). A comparison on scalability for batch big data processing on Apache Spark and Apache Flink. *Big Data Analytics*, *2*(1), 1–11. https://doi.org/10.1186/s41044-016-0020-2
4. Yu, J., Zhang, Z., & Sarwat, M. (2019). Spatial data management in apache spark: the GeoSpark perspective and beyond. *GeoInformatica*, *23*(1), 37–78. https://doi.org/10.1007/s10707-018-0330-9
5. Feng, W. (2019). Learning Apache Spark with Python. *Learning Apache Spark with Python*, 1–427.
6. Damji, J. S., Wenig, B., Das, T., & Lee, D. (2020). Learning Spark: Lightning-Fast Data Analytics. In *Learning Spark: Lightning-Fast Data Analytics*.
7. Wang, H., Li, S., Li, X., & Zhong, H. (2017). Microstructure and thermoelectric properties of doped p-type CoSb3 under TGZM effect. In *Journal of Crystal Growth* (Vol. 466). https://doi.org/10.1016/j.jcrysgro.2017.03.020
8. Xin, R., Deyhim, P., Ghodsi, A., Meng, X., & Zaharia, M. (2014). Graysort on apache spark by databricks. *GraySort Competition*, 65.
9. Pallickara, S., Science, C., & Model, C. (2015). *CS455 : Introduction to Distributed Systems [ Spring 2015 ] Dept . Of Computer Science , Colorado State University Frequently asked questions from the previous class survey S YSTEMS [ HDFS ] S LIDES C REATED B Y : S HRIDEEP P ALLICKARA CS455 : Introductio*. 1–10.
10. Assefi, M., Behravesh, E., Liu, G., & Tafti, A. P. (2017). Big data machine learning using apache spark MLlib. *Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017*, *2018*-*Janua*, 3492–3498. https://doi.org/10.1109/BigData.2017.8258338
11. Guo, R., Zhao, Y., Zou, Q., Fang, X., & Peng, S. (2018). Bioinformatics applications on Apache Spark. *GigaScience*, *7*(8), 1–10. https://doi.org/10.1093/gigascience/giy098
12. Schlicht, J. R., Livengood, B., & Shepherd, J. (1997). Development of multimedia computer applications for clinical pharmacy training. *American Journal of Pharmaceutical Education*, *61*, 287–292.
13. Wang, K., & Khan, M. M. H. (2015). Performance prediction for apache spark platform. *Proceedings - 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security and 2015 IEEE 12th International Conference on Embedded Software and Systems, H*, 166–173. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.246
14. Anderson, J. G., & Jay, S. J. (1984). The diffusion of computer applications in medical settings. *Informatics for Health and Social Care*, *9*(3–4), 251–254. https://doi.org/10.3109/14639238409015198
15. Mavridis, I., & Karatza, H. (2017). Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark. *Journal of Systems and Software*, *125*, 133–151. https://doi.org/10.1016/j.jss.2016.11.037
16. Singh, A., & Anikesh, A. (2014). Microsoft Visual Basic . Net. *International Journal of Advance Research In Science And Engineering*, *8354*(3), 210–215.
17. Kaplan, B. (1982). *Influence of Medical Values and Practices on Medical Computer Applications.* 83–88. https://doi.org/10.1007/978-1-4613-8674-2\_4
18. Belouch, M., El Hadaj, S., & Idlianmiad, M. (2018). Performance evaluation of intrusion detection based on machine learning using apache spark. *Procedia Computer Science*, *127*, 1–6. https://doi.org/10.1016/j.procs.2018.01.091
19. Safitri, N., Mubarok, M. Z., Winarko, R., Tanlega, Z., Ichlas, Z. T., Mubarok, M. Z., Magnalita, A., Vaughan, J., Sugiarto, A. T., Kaswalder, F., Cavalli, D., Paglianti, A., Sarempa, A., Isjudarto, KE, P. chao, LIU, Z. hong, Jeleni, M. N., Gumbo, J. R., Muzerengi, C., … September, M. A. K. E. Y. (2021). Name : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ c N c N c N c N. *Minerals Engineering*, *2012*(2), 3–4. https://doi.org/10.1007/s12666-019-01582-7%0Ahttps://doi.org/10.1007/s11837-020-04451-7%0Ahttp://dx.doi.org/10.1016/j.mineng.2015.10.002%0Ahttp://dx.doi.org/10.1016/j.mineng.2015.03.001%0Ahttps://doi.org/10.1016/j.mineng.2020.106488%0Ahttps://www.industry
20. Shyam R., Ganesh H.B., B., Kumar S., S., Poornachandran, P., & Soman K.P. (2015). Apache Spark a Big Data Analytics Platform for Smart Grid. *Procedia Technology*, *21*, 171–178. https://doi.org/10.1016/j.protcy.2015.10.085
21. *Applications of Computer Science in Pharmacy : An Overview Applications of Computer Science in Pharmacy : An Overview*. (2016). *July*.
22. Van Remoortere, P. (1979). Computer applications in medical care. *Mathematics and Computers in Simulation*, *21*(3), 306. https://doi.org/10.1016/0378-4754(79)90087-9
23. Belouch, M., El Hadaj, S., & Idlianmiad, M. (2018). Performance evaluation of intrusion detection based on machine learning using apache spark. *Procedia Computer Science*, *127*, 1–6. https://doi.org/10.1016/j.procs.2018.01.091
24. Shoro, G. A., & Soomro, T. R. (2015). Big Data Analysis: Apache Spark Perspective. *Global Journal of Computer Science and Technology*, *15*(1), 7–14. http://www.computerresearch.org/index.php/computer/article/viewFile/1137/1124
25. Gopalani, S., & Arora, R. (2015). Comparing Apache Spark and Map Reduce with Performance Analysis using K-Means. *International Journal of Computer Applications*, *113*(1), 8–11. https://doi.org/10.5120/19788-0531
26. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D. B., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M. J., Zadeh, R., Zaharia, M., & Talwalkar, A. (2016). MLlib: Machine learning in Apache Spark. *Journal of Machine Learning Research*, *17*, 1–7.
27. Zadeh, R. B., Meng, X., Ulanov, A., Yavuz, B., Pu, L., Venkataraman, S., Sparks, E., Staple, A., & Zaharia, M. (2016). Matrix computations and optimization in apache spark. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *13*-*17*-*Augu*, 31–38. https://doi.org/10.1145/2939672.2939675
28. Shanahan, J. G., & Dai, L. (2015). Large scale distributed data science using apache spark. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *2015*-*Augus*, 2323–2324. https://doi.org/10.1145/2783258.2789993