

DATA ENGINEERING HANDBOOK

MELİH GÜLÜM

2023

Big Data Basic	5
Büyük Veri Karakteristikleri.....	5
Büyük Veri Dosya Formatları.....	5
Avro.....	7
Parquet.....	7
ORC (Optimized Row Columnar)	8
Dosya Sıkıştırma.....	9
Apache Hadoop Giriş	10
Hadoop Temel Özellikleri.....	10
Hadoop Ecosystem.....	12
Apache Hive Giriş	17
Partitions ve Buckets	18
Hive internal ve external table	19
Apache Sqoop Giriş	19
Apache Kafka Giriş.....	20
Message Delivery Reliability.....	21
Kafka'nın Temel Kavramları ve Mimarisi.....	22
Big Data Processing with Apache Spark.....	24
Apache Spark Giriş	24
Neden Spark?.....	25
MapReduce ile Farklılıklar.....	25
Spark Stack.....	26
Spark Çalışma Modları.....	27
Spark Mimarisi	29
Dataframe API Giriş.....	30
Structed (Yapısal) Nedir?	30
Spark Structed API.....	30
Schema	30
Spark Datafame.....	31
Spark Datasets	31
Spark Structed API Çalışma Planı.....	31
Lazy Evaluation, Transformations and Action.....	33
Spark Optimizasyon Teknikleri	35
Data Serialization.....	35
Cache and Persist	35

Data Structure Tuning	36
Garbage Collection Optimizasyonu	36
API Seçimi.....	36
UDF Kullanmaktan Kaçınmak	37
Shuffle Kullanmaktan Kaçının	37
Spark Join Strategies	37
Delta Lake	42
Spark Machine Learning	44
Spark ML Pipelines.....	44
Spark ML Temel Kavramlar	44
Real-Time Data Processing with Spark	45
Spark Structured Streaming Giriş	45
Spark Structured Streaming Programming Model.....	45
Basic Concept	45
Handling Event-time and Late Data.....	47
Fault Tolerance.....	47
API using Datasets and DataFrames	48
Creating streaming DataFrames and streaming Datasets	48
Operations on streaming DataFrames/Datasets	49
Window Operations on Event Time.....	49
Handling Late Data and Watermarking	49
Join Operations	50
Streaming Deduplication	50
Starting Streaming Queries	51
NoSQL	53
CAP Teoremi	53
CAP theorem NoSQL database types.....	54
NoSQL Veritabanlarına Giriş.....	54
RDBMS ACID.....	54
NoSQL BASE.....	55
NoSQL Veritabanlarını Sınıflandırma	55
Wide Column: Apache Cassandra.....	57
Document Bases: MongoDB	60
Full-Text Search: Elasticsearch	63
Data Pipelines and Workflow Scheduling.....	65
Apache Airflow.....	65

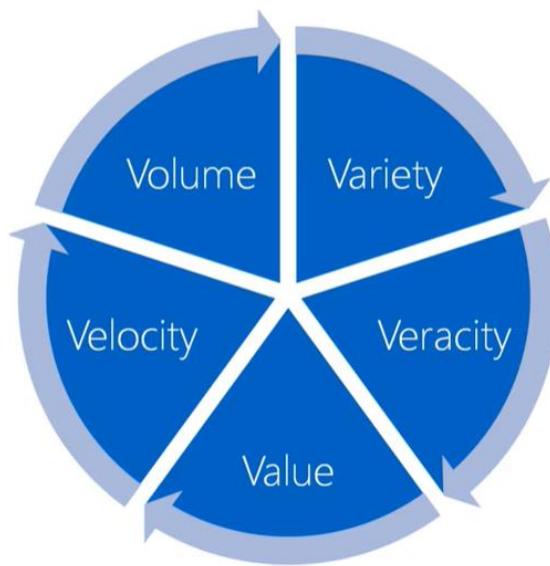
Neden Airflow?.....	65
Airflow'un Faydaları.....	65
Airflow Architecture	65
Logstash	67
Logstash Pipeline	67
Apache NiFi.....	67
Apache NiFi Temel Bileşenleri.....	68
Apache NiFi Architecture.....	68
Container Environments and Deploying Machine Learning Models.....	70
MLOPS	70
MLOPS Maturity Model	70
MLFlow.....	71
Core Components of MLflow.....	72
Docker	72
Docker Mimarisi	72
Docker'ın Avantajları ve Dezavantajları	73
Kubernetes.....	74
Kubernetes Ne Yapar?	74
Kubernetes Cluster Mimarisi	75

Big Data Basic

- Büyük veri:
 - Geleneksel teknoloji, işleme yöntemleri, algoritma, mimari yapı (scale-up, scale-out) ve çerçevede işlenmesi, saklanması mümkün olmayan;
 - Veri işlemek, saklamak ve yönetmek için yeni teknoloji, platform ve yaklaşılara ihtiyaç duyan;
 - Hızla çoğalan, karmaşıklığı ve belirsizliği yüksek, çok çeşitli kaynaklardan farklı formatlarda elde edilen;
 - Bu tür veriden bir çok fayda, değer, fırsat elde edilmesini olanaklı kıلان büyük hacimli verilerdir.

Büyük Veri Karakteristikleri

- **Volume** (Hacim): Çok büyük boyutlarda olması (TB, PB ve üstü)
- **Variety** (Çeşitlilik): Çok farklı kaynaktan türemesi, farklı formatta olması
- **Veracity** (Belirsizlik Derecesi): Doğruluğun kontrole ihtiyacı
- **Value** (Değer): Bu kadar çok veriden değer üretebilme
- **Velocity** (Hızlı Çoğalma): Mevcut verinin %80'i son 2-3 yılda üretildi.



Büyük Veri Dosya Formatları

- Verileri data lake'de nasıl depoladığınız kritik öneme sahiptir ve formatı, sıkıştırmayı ve özellikle verilerinizi nasıl böülümlendirdiğinizi dikkate almanız gereklidir.
- En yaygın formatlar:
 - CSV

- JSON
- AVRO
- Protocol Buffers
- Parquet
- ORC

Property	Avro	Parquet	ORC
Schema Evolution	Full	Costly	Limited to addition
Compression	Less efficient	Best snappy	Best zlib
Splitability	Partial	Partial	Full
Data storage orientation	Row	Columnar	Columnar
Optimized for	Write	Read	Read
Predicate Pushdown	No	Okayish	Efficient
ACID		Non compliant	Compliant
Best use case	Kafka, Druid	Spark, Arrow	Hive & Presto

- Verilerinizi düzenlemenin iki ana yolu vardır: satırlar ve sütunlar. Hangisini seçeceğiniz, verilerinizi ne kadar verimli bir şekilde depoladığınızı ve sorguladığınızı büyük ölçüde kontrol eder.
 - **Rows:** veriler kayıtlara göre düzenlenir. Bunu, verileri organize etmenin ve yönetmenin daha "geleneksel" bir yolu olarak düşünebiliriz.
 - 1, Michael, Jones, Dallas, 32
 - 2, Preston, James, Boston, 25
 - **Columns:** her tablo sütununun (alanının) değerleri yan yana saklanır. Bu, benzer öğelerin gruplandırıldığı ve yan yana depolandığı anlamına gelir. Alanlar içinde okuma sırası korunur; bu, verileri kayıtlara bağlama yeteneğini korur.
 - ID: 1, 2
 - First Name: Michael, Preston
 - Last Name: Jones, James
 - City: Dallas, Boston
 - Age: 32, 25

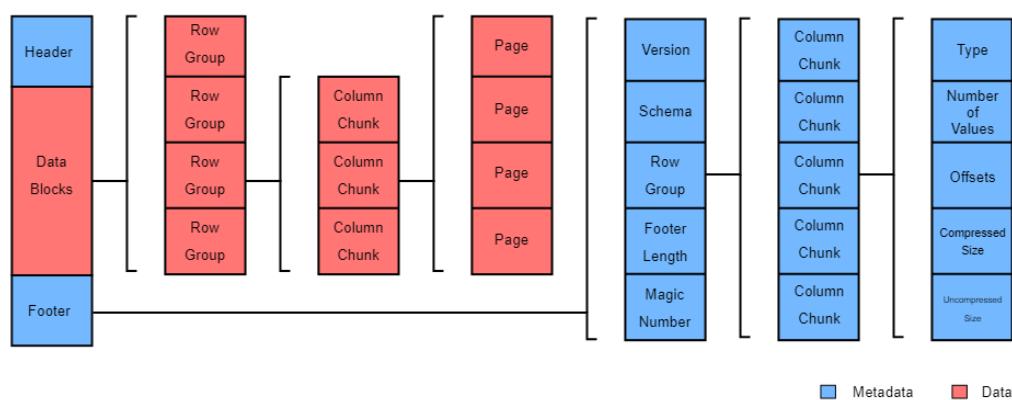
	A	B	C	D	E
1	ID	First Name	Last Name	City	Age
2	1	Michael	Jones	Dallas	32
3	2	Preston	James	Boston	25

Avro

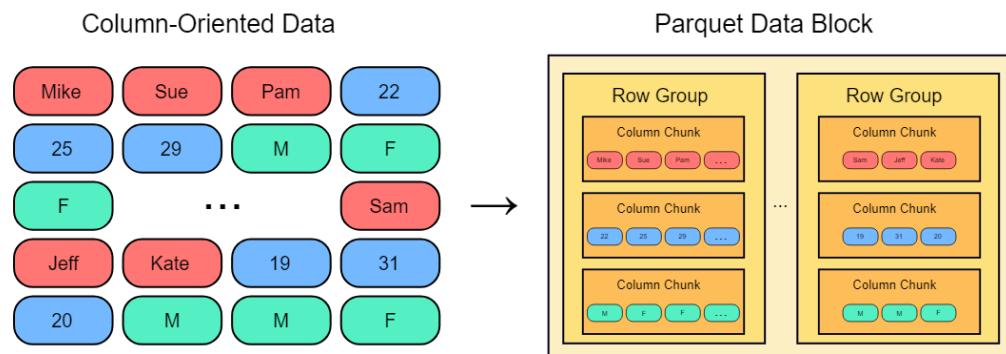
- Apache Kafka için sıkılıkla önerilen açık kaynaklı bir formattır.
- Hadoop'ta veri serileştirilirken tercih edilir.
- Avro formatı, veri tanımlarını (şema) JSON'da saklar ve kolayca okunup yorumlanır.
- Dosya içindeki veriler binary biçimde depolanır, bu da onu kompakt ve yerden tasarruflu hale getirir.
- Avro formatlı dosyalar splittable ve compressible (ancak bazı sütunlu veriler kadar sıkıştırılmazlar).

Parquet

- Parke dosyaları, karmaşık iç içe geçmiş veri yapılarını düz formatta destekler ve çoklu sıkıştırma seçenekleri sunar.
- Veriler sütunlarda saklandığından yüksek oranda sıkıştırılabilir (sıkıştırma algoritmaları genellikle sütunlarda yer alan düşük entropili bilgi içeren verilerle daha iyi çalışır) ve ayrılabilirler.
- Parquet dosyaları, dosyanın blok sınırlarına ilişkin bilgileri içeren dosya altbilgi meta verilerini depoladıkları için bölünebilir.
- Parquet aynı zamanda self-describe'tır. Dosya şemasını ve yapısını içeren meta verileri içerir. Bunu, Parquet dosyalarını yazmak, saklamak ve okumak için farklı hizmetleri ayırmak için kullanılır.
- Her Parquet dosyasında bir **header**, bir veya daha fazla **data block** ve bir **footer** bulunur. Bu bileşenler içinde bir Parquet dosyası iki farklı türde bilgiyi saklar: meta veriler ve veriler. Spesifik olarak, meta veriler header ve footer depolanırken veriler data block depolanır.



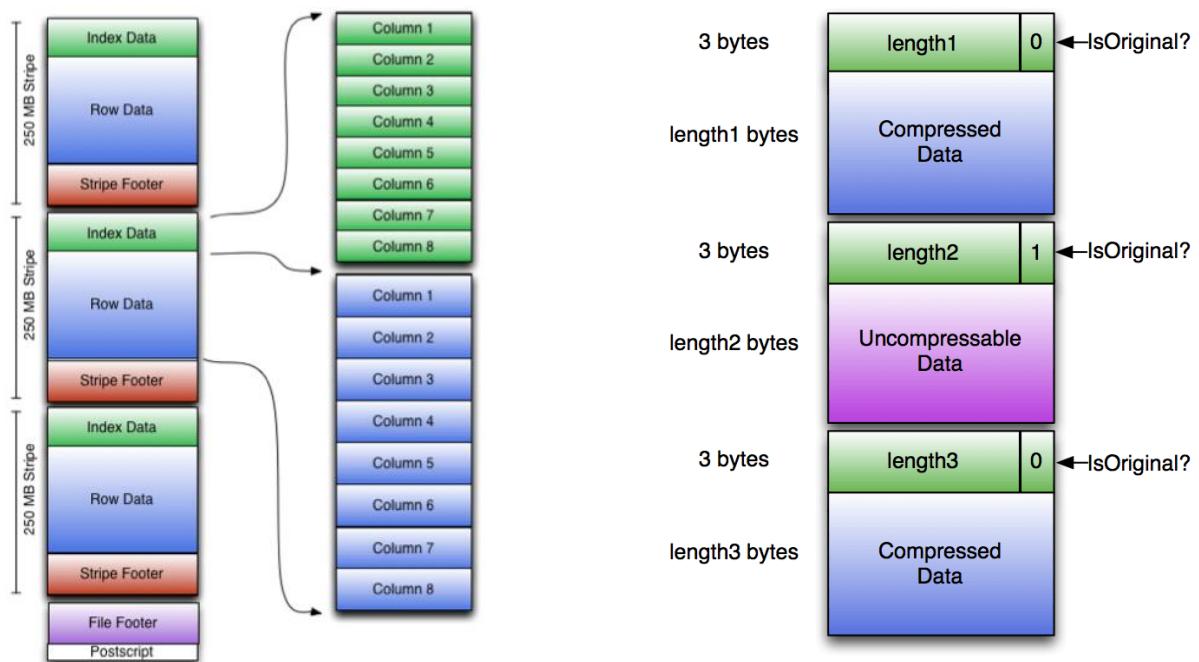
- Başlık, dosyasının Parquet formatında olduğunu temsil eden başlıkta 4 baytlik magic number biçiminde meta veriler içerir. Dosyaya ilgili kalan meta veriler footer bölümünde saklanır. Hakkında meta veriler içerir:
 - Row Groups
 - Columns
 - Parquet Versiyonu
 - 4 Bitlik Magic Number
- Bir Parquet dosyasında her veri bloğu, satır gruplarının bir koleksiyonu olarak depolanır. Bu satır grupları, sütun parçalarının bir koleksiyonu olarak depolanır. Bir satır grubu bir dizi satıra karşılık gelirken, bir sütun öbeği veri kümesindeki tek bir sütuna karşılık gelir. Sütun parçalarındaki veriler, sütun değerlerine karşılık gelen sayfalar halinde düzenlenir.



ORC (Optimized Row Columnar)

- ORC, Apache Hive verilerini depolamak için oldukça verimli bir yol sağlar.
- Diğer dosya formatlarının sınırlamalarının üstesinden gelmek için tasarlanmıştır.
- ORC dosya formatı, satır koleksiyonlarını tek bir dosyada, dosya içinde sütunlu bir formatta saklar. Bu, bir küme genelinde satır koleksiyonlarının paralel işlenmesine olanak tanır. Sütunlu düzen nedeniyle, her dosya sıkıştırma için idealdir; okuma ve sıkıştırmayı açma yüklerini azaltmak için verilerin ve sütunların atlanması olanak tanır.
- Bir ORC dosyası, bir dosya footer'da stripes adı verilen satır verileri gruplarını ve yardımcı bilgileri içerir. Dosyanın sonunda bir postscript, sıkıştırma parametrelerini ve sıkıştırılmış altbilginin boyutunu tutar.

- Varsayılan şerit boyutu 250 MB'dir. Büyük şerit boyutları, HDFS'den büyük, verimli okumalara olanak tanır.



Dosya Sıkıştırma

- Dosya boyutu ile CPU maliyetleri arasındaki dengeyi göz önünde bulundurarak verileri nasıl sıkıştıracağınızı da düşünmeniz gerekir. Bazı sıkıştırma algoritmaları daha hızlıdır ancak daha büyük dosya boyutuna sahiptir, bazıları ise daha yavaş ancak daha iyi sıkıştırma oranlarına sahiptir.

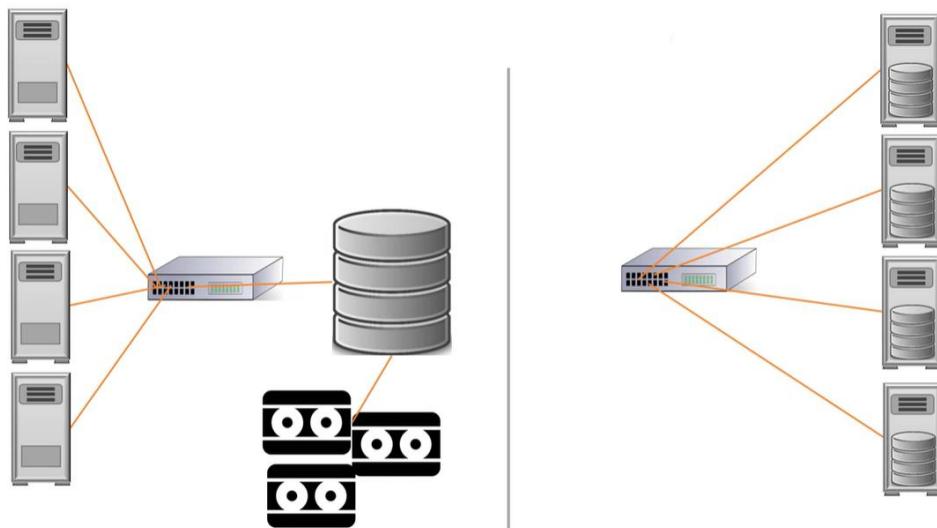
Format	Algorithm	Strategy	Emphasis	Comments
zlib	Uses DEFLATE (LZ77 and Huffman coding)	Dictionary-based, API	Compression ratio	Default codec
gzip	Wrapper around zlib	Dictionary-based, standard compression utility	Same as zlib, codec operates on and produces standard gzip files	For data interchange on and off Hadoop
bzip2	Burrows-Wheeler transform	Transform-based, block-oriented	Higher compression ratios than zlib	Common for Pig
LZO	Variant of LZ77	Dictionary-based, block-oriented, API	High compression speeds	Common for intermediate compression, HBase tables
LZ4	Simplified variant of LZ77	Fast scan, API	Very high compression speeds	Available in newer Hadoop distributions
Snappy	LZ77	Block-oriented, API	Very high compression speeds	Came out of Google, previously known as Zippy

Apache Hadoop Giriş

- Apache Hadoop bir yazılım kütüphanesidir, bir framework'tür. Bu framework; çok büyük hacimli veri setlerinin sunucu kümeleri (clusters) üzerinde basit programlama modelleriyle dağıtık şekilde işlenmesine imkan verir.

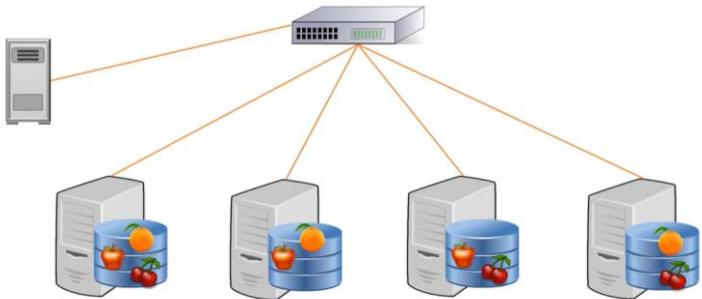
Hadoop Temel Özellikleri

- Açık kaynaklı olması
- Yazılım platformu olması
- Bilgisayar kümeleri (cluster) üzerinde çalışabilmesi
- Veri depolama
- Dağıtık hesaplama
- Büyük hacimli veriyi işlemesi
- **Peki niçin Hadoop veya Hadoop'un mimarisi klasik neden farklı diyecek olursak:**
 - Solda gördüğümüz klasik veri mimarisidir. Sunucular, Anahtar, Enterprise Storage ve Yedekleme'den oluşur. Bu mimaride sunucular üzerinde ciddi veri tutulmaz. Hepsi Enterprise Storage üzerinde tutulur. Sunucu gittiği zaman veri kaybı olmaz sadece hizmete erişim sıkıntısı uğrar.
 - Sağda ise, Hadoop mimarisinde veriler genelde sunucular üzerindedir. Enterprise Storage ve yedeklemeye ihtiyaç yoktur. Çünkü replikasyon vardır.



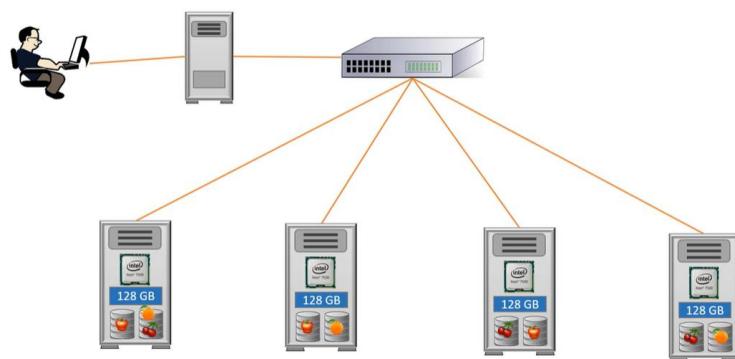
- **Hadoop nasıl hataya karşı daha dayanıklıdır?**
 - Dayanaklı dediğimiz nokta, çalışan bir sunucun başına bir iş geldiğinde **hizmet kesintisi** ugramıyor ve veri kaybı yaşanmıyor.

- Diyelim ki 4. Sunucu kapandı. Master Sunucu, 4. Sunucudaki verileri kopyalayıp diğerlerine gönderiyor.



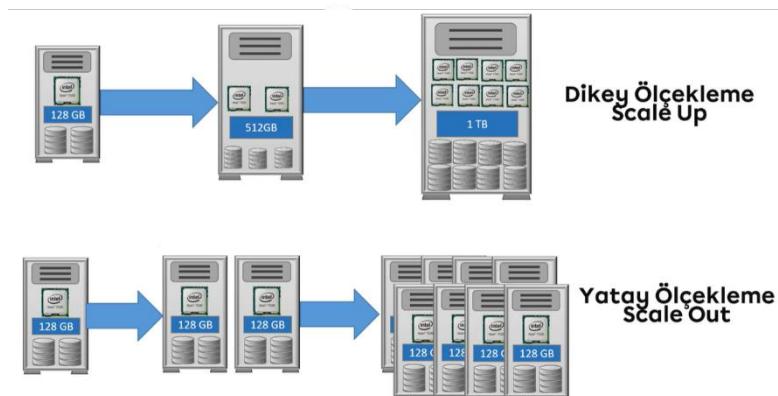
- Dağıtık İşleme nasıl oluyor?**

- Büyük veri mantığında, veri sorgulayana gitmez, sorgu veriye gider. Çünkü bu kadar büyük verinin bir yerde toplanması mümkün değil. Sadece sonuçlar kullanıcuya gider.

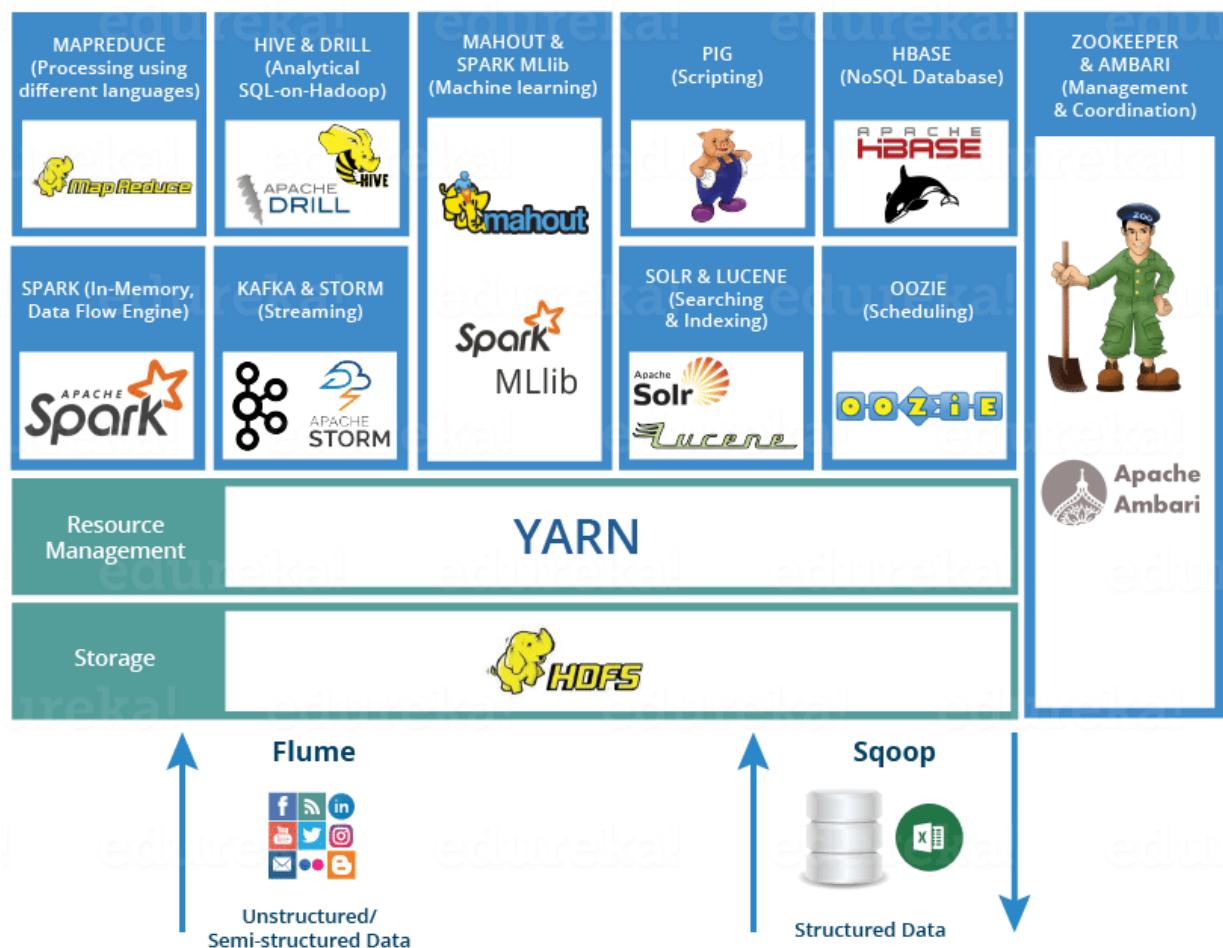
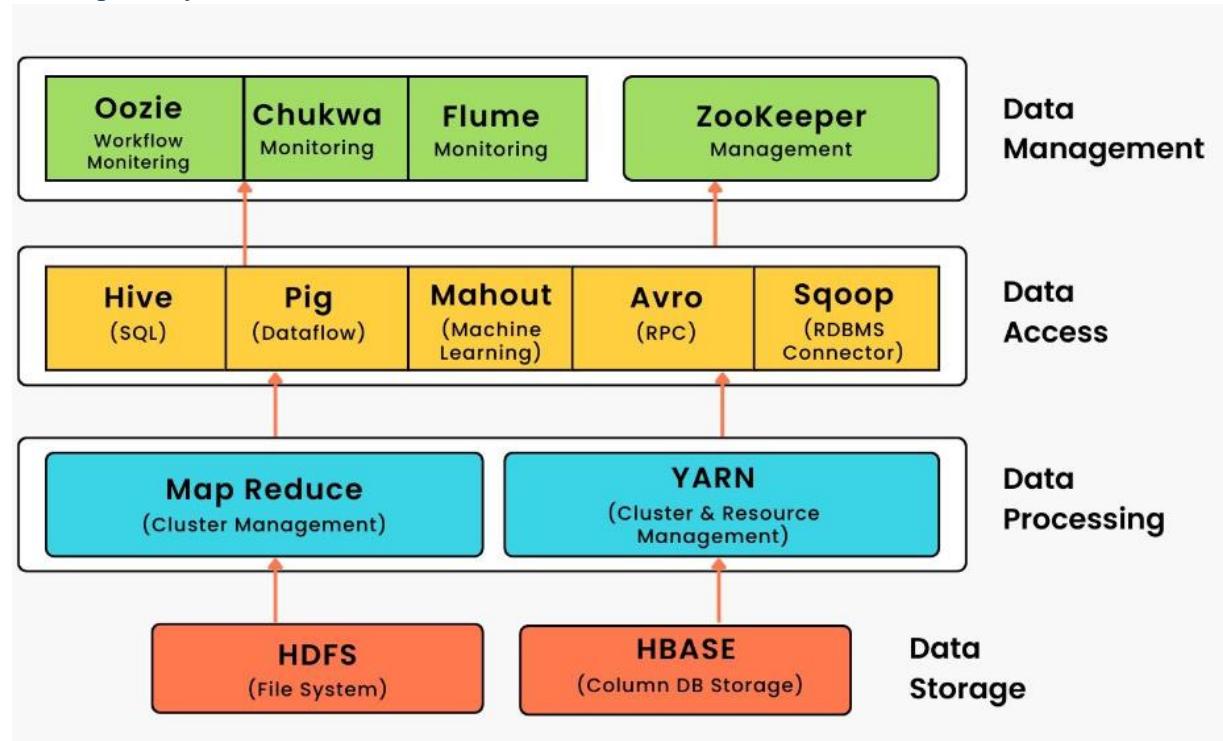


- Ölçekleme**

- İki tür ölçekleme vardır.
- Scale-Up klasik teknolojilerin ölçeklemesidir. Kaynaklar az geldikçe artmaktadır. Ama bunun bir sınırı var çünkü bir makineye ilave edeceğimiz kaynak sınırlıdır.
- Scale-Out ise sıradan makineleri beraber kullanmaktadır. İhtiyaç kadar eklenir ve devam edilir.



Hadoop Ecosystem



HDFS (Hadoop Distributed File System)

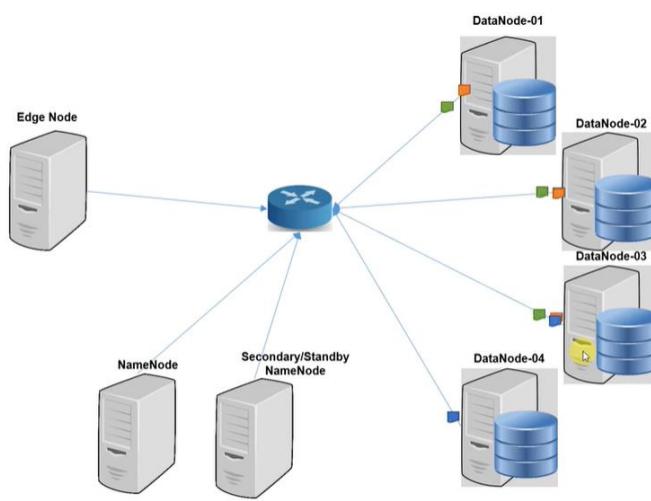
- HDFS çok büyük hacimli verileri depolamak için java tabanlı dağıtık bir dosya sistemidir.
- Hatalara karşı dayanıklıdır.
- Ölçeklenebilir.
- Düşük maliyetlidir.
- Büyük veri için idealdir.

HDFS nasıl çalışır?

- Resimde bir Edge Sunucusu, iki tane Master Sunucusu (NameNode) ve Worker Sunucuları (DataNode) görülmektedir. Worker Sunucuların görevi veriyi saklamak, işlemek ve Master Sunucusu'ların talimatlarını yerine getirmektir.
- Gelen büyük veriler parçalı bir şekilde Worker Sunucular üzerinde tutuluyor ve bu şekilde işleniyor.

NameNode Ve DataNode

- NameNode'un başlıca görevleri:
 - Verinin metadata'sının, blok adreslerinin ve dosya sistemlerinin tutulduğu yerdir.
 - Kullanıcı erişim yetki kontrolü yaparlar.
 - Dosya sistemi operasyonlarını yönetirler (okuma, yazma, yaratma, taşıma vs.).
 - DataNode'ları kayıt ederler. Canlı olup olmadıklarını kontrol ederler.
 - Replikasyon talimatı verirler.
 - DataNode'lardan gelen blok raporlarını incelemek.
- DataNode'ların görevi ise:
 - Veriyi tutmak, işlemek ve varlığı konusunda NameNode'lara rapor vermekti.



HDFS Erişim Yöntemleri

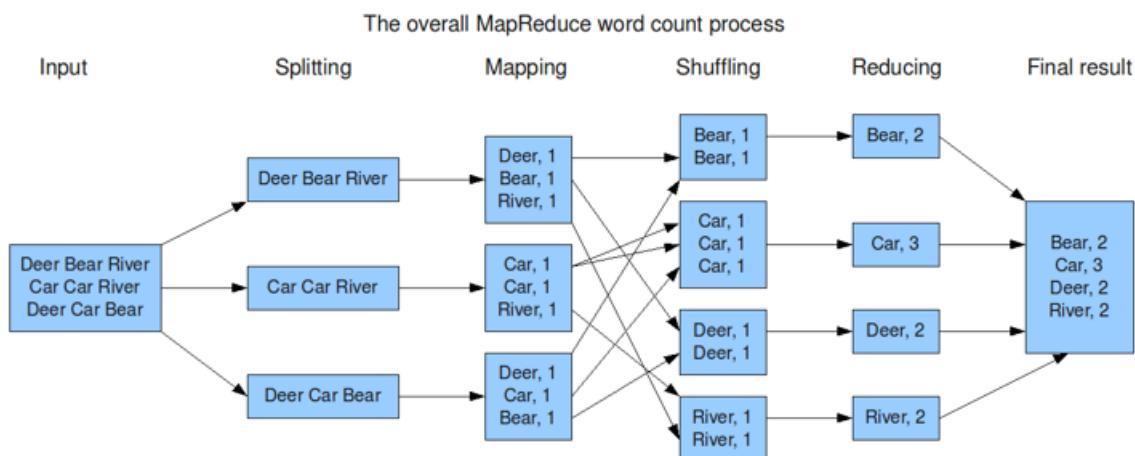
- Web arayüzü
- Komut Satırı
- HTTPS/HDFS vekil sunucuları (Proxies)
- Java Interface
- NFC Gateway (legacy app'lerin entegrasyonu haricinde pek tercih edilmeyen performans, güvenlik ve stabilite bakımından tatmin edici olmayan yöntemdir.)

MapReduce

- HDFS ve YARN ile beraber Hadoop'un en temel bileşeninden biridir.
- Büyük veri işlemek için bir framework'tür.
- Google'un web sayfalarını indeksleme ihtiyacından doğdu.
- Java tabanlı çalışmaktadır.
- MapReduce mantığı anlaşılsa büyük verinin de mantığı anlaşılmış olur.
- **Map()** parçala ve hesaplanacak hale dönüştür demektir.
- **Reduce()** ise parçaları hesapla ve sonuçları getir demektir.

MapReduce Temel Özellikleri

- Sunucu kümeleri üzerinde dağıtık ve paralel çalışabilmektedir.
- Ölçeklenebilir.
- Düşük maliyetli.
- Büyük veri için ideal.
- Hatalara karşı dayanıklıdır.



YARN

- Yet Another Resource Negotiator anlamına gelir.
- Hadoop'u MapReduce'a mahkum olmaktan kurtarır.
- YARN ile Hadoop gerçek bir ölçülebilir büyük veri işleme platformu oldu.
- YARN dünyasında bir miktar belleğin ve birkaç çekirdeğin oluşturduğu kavrama konteynır denir.
- YARN'ın çözmeye çalıştığı problem, uygulamaların neye göre, nasıl ve hangi öncelikle bu kaynaklara tahsis edileceğine dair bir çözüm sağlamaya çalışır.
 - YARN olmazsa bir süre sonra işler karışabilir. Öncelik problemi vs. yaşanabilir.
- YARN;
 - Kaynak tasarrufu (Veri bilinci)
 - Güvenlik
 - Randıman
 - Öncelik
 - Hakkaniyet
 - Çeşitlilik sağlar.

YARN'in En Temel Üç Bileşeni

1. ResourceManager (RM) → Cluster için
 2. NodeManager (NM) → Her bir Node için
 3. ApplicationMaster (AM) → Her bir uygulama için
- RM ve NM kalıcıdır. Cluster çalıştığı sürece vardır. AM ise bir uygulama çalıştığı sürece vardır ve daha sonra kaybolur.

ResourceManager

- Kaynak tahsisinde nihai otoritedir. Merkezde Master Sunucusu'da yer alır.
- Cluster başında bir bakışı vardır.
- Scheduling öncelikleri ve mevcut kaynaklara göre taleplere cevap verir.
- İki temel bileşeni vardır:
 - Scheduler
 - Application Manager

NodeManager

- Her bir işçi sunucusu üzerinde olur ve

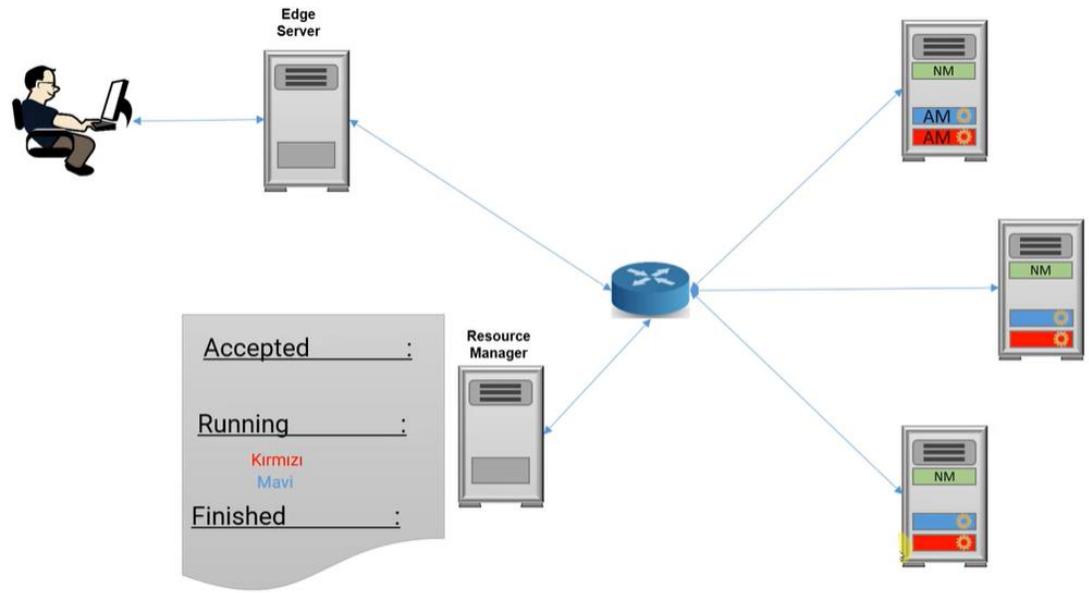
- Nabız, kaynak takibi, Hata raporu, konteynır hayat döngüsü, Log aggregation gibi işleri görmektedir.
- ResourceManager'ın işçi sunucudaki gözü kulağıdır.
- Kalıcı bir servis olarak çalışır.
- En temel görevi konteynır yönetimidir.

ApplicationMaster

- Her bir uygulama için bir tanedir.
- İşçi sunucuda çalışır.
- Kalıcı servis değildir. Geçici konteynır içinde çalışır.
- YARN'dan kabul alırsa var olabilir.
 - Yani RM'den kaynak alabilirse var olur.
- Uygulamanın çalışması için kaynakları NodeManager(lar) ve ResourceManager ile koordine eder.

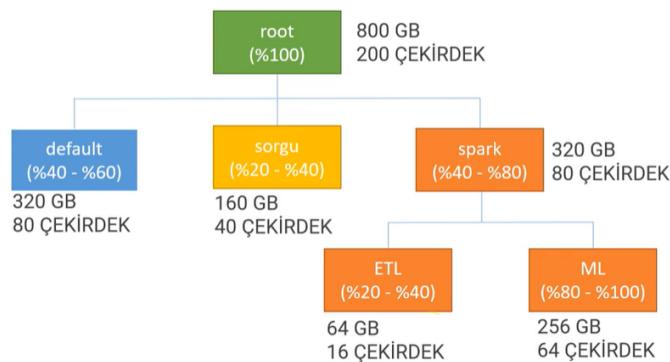
Bir Uygulamanın YARN Yolculuğu

- Kullanıcı bir uygulama gönderdiğini varsayıalım ve bu uygulama “mavi” olsun.
- “Mavi” uygulama ResourceManager'a geldi. RM çeşitli kontroller sağlıyor.
 - Yetkisi var mı?
 - Tarife (Scheduler) Önceliği
 - İstediği kadar kaynağım var mı?
 - İşlemek istediği veri nerede?
- Daha sonra YARN'ın temel olarak üç durumu vardır.
 - Eğer kontrolleri geçerse, uygulamayı **Accepted**'a alıyor.
 - Her şey müsaitse uygulamalar **Running**'e geçer.
 - AM için bir konteynır verir ve AM kaç konteynira ihtiyacı olduğunu talep eder. Kaynakların uygunluk durumuna göre RM, NodeManager'a kaynak tahsisi için görev verir.
 - Uygulamanın işi biterse **Finished** olur ve kullandığı kaynakları boşaltır.



Capacity Scheduler

- Kuyruğun asgari kaynak tahsisini garantiye,
- Kuyruğun azami kaynak kullanımı sınırlama,
- Kullanıcıların kaynaktan alabilecekleri asgari kaynak tahsisini garantiye,
- Kullanıcıların kuyruktan alabilecekleri azami kaynak kullanımı sınırlama,
- Kuyruğun durumunu kontrol etme,
- Kuyruk erişim kontrolü sağlamsaktadır.



Apache Hive Giriş

- Neden Hive?
 - Onlarca satır MapReduce kodu ile yapılan iş, bir satır Hive sorgusu ile yapılabilir.
- SQL + MapReduce = Hive

- Hive, HDFS üzerinde saklanan verileri SQL benzeri bir dil ile sorgulanmasına olanak sağlar.
- Facebook yapmıştır. Daha sonra Apache topluluğuna devredilmiştir.
- Tablo içinde saklanabilen yapısal veriyi işler.
- Gerçek zamanlı (akan veri) değil, daha çok batch processing için var.
- MapReduce ile HDFS üzerinde yapılan veri manipülasyonun SQL kullanarak yapılmasına olanak sağlar.

Hive Ne değildir?

- Hive bir veri tabanı değildir.
- Operasyonel veri tabanı ihtiyacını karşılamaz. Daha çok analitik büyük sorgular ve veri ambarı ihtiyaçlarına odaklanır.
- Satır bazlı insert, update, delete için uygun değildir (son sürümlerde desteklenmesine rağmen oldukça yavaş).
- İnteraktif sorgular için de uygun değildir. Cevap için makul bir süre bekletir. Çünkü sorguyu önce MapReduce veya tez koduna çevirir sonra operasyona başlar.

Partitions ve Buckets

- Her ikisi de veri organizasyonu ile kavramlardır.
- **Partitions**, verinin kategorik bir sütun bazında farklı klasörlerde toplanmasıdır.
 - Aynı kategoriler bir arada olur. Sorgu hızı artar.
- **Buckets**, klasör değil dosyadır. Birbirine yakın olan sütunları bir arada saklamaktır. Aynı partitions altında birden fazla klasör olabilir.
 - Kendi başına Partitioning ile birlikte kullanılabilir.
 - Hangi kaydın hangi Bucket içinde depolanacağına hashing algoritması tarafından belirlenir.
 - Bucket sayısını tablo oluştururken belirleyebiliriz.
 - Bucketed Map Joins hızlıdır. (İki Bucket tablo çok hızlı birleşir.)

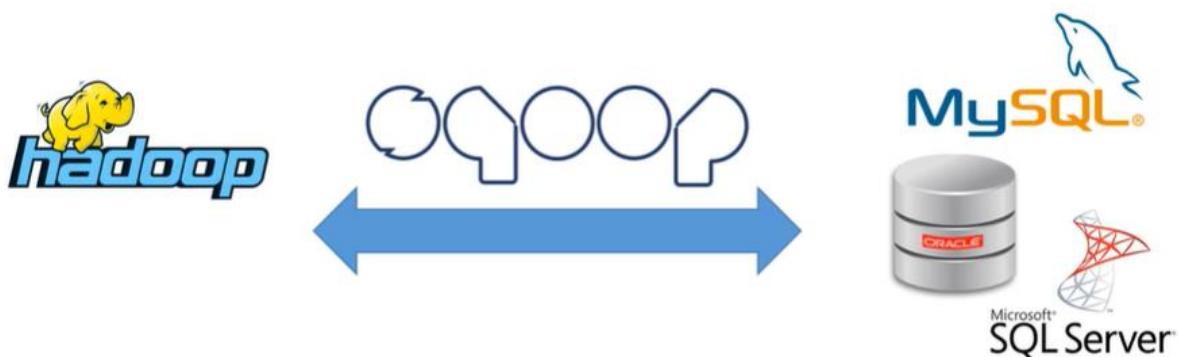


Hive internal ve external table

- Farklı metadata'ların nasıl tutulduğuna göre değişir.
- **Internal** tablo için hem metadata hem gerçek data hive tarafından yönetilir ve saklanırken **External** tabloda sadece metadata hive tarafından tutulur.
- Eğer Internal tablo drop edilirse hem metadata hem de veri silinir. Ancak External tablo silinirse sadece metadata silinir, veri HDFS'te kalır.
- Create Table komutunda aksi belirtilmemezse varsayılan olarak Hive Internal Table yaratır.

Apache Sqoop Giriş

- Halihazırda var olan operasyonel verilerin tutıldığı ilişkisel veri tabanları var. Sqoop, Hadoop ile bu veri tabanları arasındaki veri aktarım ihtiyacını karşılamaktadır.



- Sqoop Eval: SQL ifadesini değerlendirir ve sonucu gösterir.
- Sqoop Import: İlişkisel veri tabanından Hadoop'a tablo yükler.
- Sqoop Export: HDFS dosyasını ilişkisel veri tabanına çıkartır.

Apache Kafka Giriş

- Kafka kendisini **Distributed Streaming Platform** olarak tanımlamaktadır.
- Yayınla (Publish) ve Abone Ol (Subscribe) mantığıyla çalışır.
- Mesajları hataya dayanıklı bir şekilde belirli bir süre saklar.
- Kendisine bırakılan mesajı olduğu gibi iletir, değiştirmez.
- Cluster halinde çalışır. Yatay ölçeklenebilir.
- Linkedin tarafından geliştirildi. Daha sonra açık kaynaklı bir proje oldu.
- Destekçi firması Confluent'tir.
- 100 node gibi rakamlara kadar ölçeklenebilir.
- Saniyede milyonlarca mesaj işleyebilir.
- Yüksek performanslıdır.
- Gecikme süresi milisaniyedir. Bu sebeple realtime'dır.
- Çok yaygın bir kullanımına sahiptir.
- Fortune 500'ün neredeyse yarısını kullanabiliyor.

Neden Kafka?

- Eğer 5 farklı kaynak, 7 farklı hedef olsa 35 farklı veri entegrasyonu yapılması gereklidir.
- Bunlar için farklı bağlantılar kullanılabilir (HTTP, TCP, REST, FRP, JDBC, ODBC ...)
- Formatları farklı olabilir (Binary, json, csv, avro, parquet)
- En önemlisi, verinin kaynağının üretim hızı, hedef kaynağın tüketim hızından çok fazla olabilir.
 - Bu lag'e sebep olabilir. Bu birikmeyi istemeyiz.
- **Bu nedenle veriler direkt olarak değil, Kafka (asynchronous transmission) gibi bir sistem üzerinden aktarılıyor.**

Ne için Kullanılıyor?

- Mesajlaşma sistemi olarak
- Farklı lokasyonlardan metrikleri toplamak için
- Faaliyet izleme (activity tracking)
- Uygulama log'larının toplanması için
- Spark veya Flink ile beraber akan veri işlemek için
- Hadoop ekosistemi ile uyumlu, tüm Hadoop şirketleri tarafından destekleniyor.

Message Delivery Reliability

- Mesaj, Kafka'nın işlediği en küçük parçadır. Bir tablo veya metin dosyasının bir satırı olarak örnek verilebilir.
- Bir mesaja erişmek istersek Topic, Partition ve Offset'ın bilinmesi gereklidir.

At-most-once

- En fazla bir mesaj ulaşılır.
- Mesaj kopyalanmaz.
- Mükerrer mesaj olmaz.
- Mesaj kaybolabilir.
- Bu durum performansın istendiği ancak veri kaybının umursanmadığı durumlarda kullanılabilir.

At-least-once

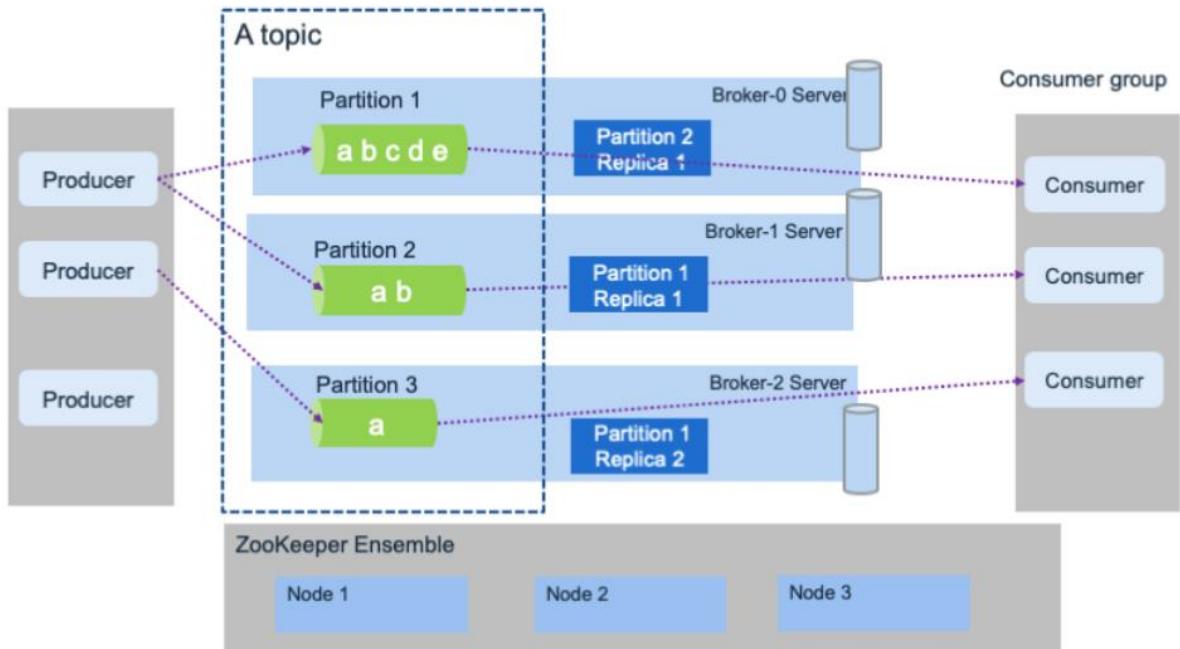
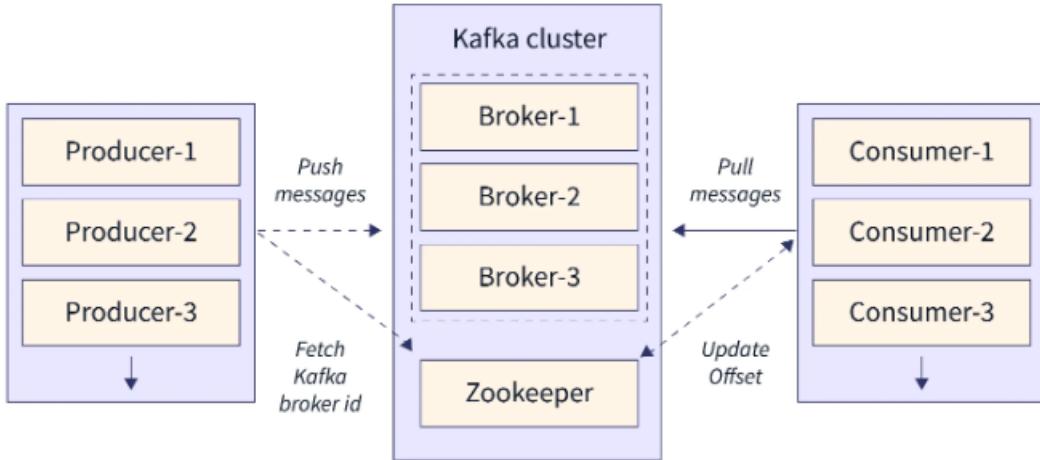
- En az bir kez mesaj ulaşır.
- Mesaj kopyalanabilir.
- Mükerrer mesaj olabilir.
- Mesaj kaybolmaz.

Exactly-once

- Mesaj kopyalanmaz.
- Mesaj kaybolmaz.
- Mükerrer mesaj olmaz.
- Mekanizmaya verilen her mesaj alıcıya tam bir teslimat yapılmasına anlamına gelir.
- Diğerlerine göre biraz daha maliyetlidir.

Kafka'nın Temel Kavramları ve Mimarisi

Kafka Architecture



- Producer
 - Kafka'ya mesaj gönderen uygulamadır.
- Consumer
 - Kafka'ya gelip mesajı okuyan uygulamadır.
- Broker
 - Cluster içinde Kafka sunucusuna verilen addır.
- Cluster
- Topic
 - İlgili mesajların bir arada toplandığı bir yapıdır.

- Topic için tablo, mesaj için satır benzetmesi yapılabilir.
- Partitions
 - Topic içerisindeki fiziksel veri parçalarıdır.
 - Performans için tercih ediliyor.
 - Parça sayısını topic bazında producer olarak belirleyebiliyoruz.
 - Hataya karşı dayanıklılığı sağlar.
 - Replikasyona imkan sağlar.
 - Replikasyon topic seviyesinde belirlenir ama Partitions'a uygulanır.
 - Replikasyon Factor'ü Brooker sayısından fazla olamaz.
- Offset
 - Topic içerisindeki parçaların içindeki mesajların sıra numarasına denir.
- Consumer Group

Big Data Processing with Apache Spark

Apache Spark Giriş

- Bütünleşik bir hesaplama motorudur (framework).
 - Bütünleşik demek ise, büyük veri uygulamaları için her türlü ihtiyacı karşılayacak şekilde tasarlanmış ve geliştirilmiş olmalıdır.
 - Bu ihtiyaçlar ise, Veri okuma/ yazma, programlama, SQL, Makine Öğrenmesi, streaming, graf analizi, Python, scala, Java, R ... hepsi aynı çatı altında ve birbirleri ile uyumlu API setleri içindedir.
- Bilgisayar kümelerinde paralel veri işleme için bir dizi kütüphane oluşturmaktadır.
- Büyük veri ile uğraşan veri bilimci, veri mühendisi, makine öğrenimi mühendisi vb. için birçok araç, API ve programlama dili sunar.
- Spark'ın en büyük iddiası gerçek dünyadan büyük veri analitiği problemlerine çözüm olmaktadır.
 - İnteraktif veri analizi yapılabilir.
 - IDE ile canlı ortamda uzun soluklu uygulama, ETL, veri analizi uygulamaları geliştirme yapılabilir.
- Spark bütünlesiktir. Birçok aracı barındırır ama sınırlarını da iyi çizer. **Veri depolamaz.** Veriyi olduğu yerde analiz eder. Gerektiği kadarını Spark Cluster belleğine çeker.

Büyük veri de neden Spark'a veya dağıtık işlemeye ihtiyaç vardır?

- Bir tane disk için ortalama hızlar aşağıdaki gibi olsun. Eğer disk sayısı artar ise doğrusal olarak okuma/yazma hızı da artmaktadır. Büyük veriyi 50 parçaaya böldüğümüzü varsayırsak bu süre çok daha kısalacaktır.

Ortalama 100 MB/s hız ile
10 sn. → 1 Gb
1 dk. → 6 Gb
10 dk. → 60 Gb
1 saat → 360 Gb
3 saat ~ 1 Tb

Ortalama 400 MB/s hız ile
1,25 sn. → 1 Gb
7,5 sn. → 6 Gb
1,25 dk. → 60 Gb
7,5 dk. → 360 Gb
22 dk. ~ 1 Tb



- Spark diskleri bu büyük verileri parçalar halinde dağıtmaktadır ve birçok bilgisayarın donanımını kullanmaktadır. Bütün bunları birleştirerek büyük veriyi etkin bir şekilde işleyebilmektedir.

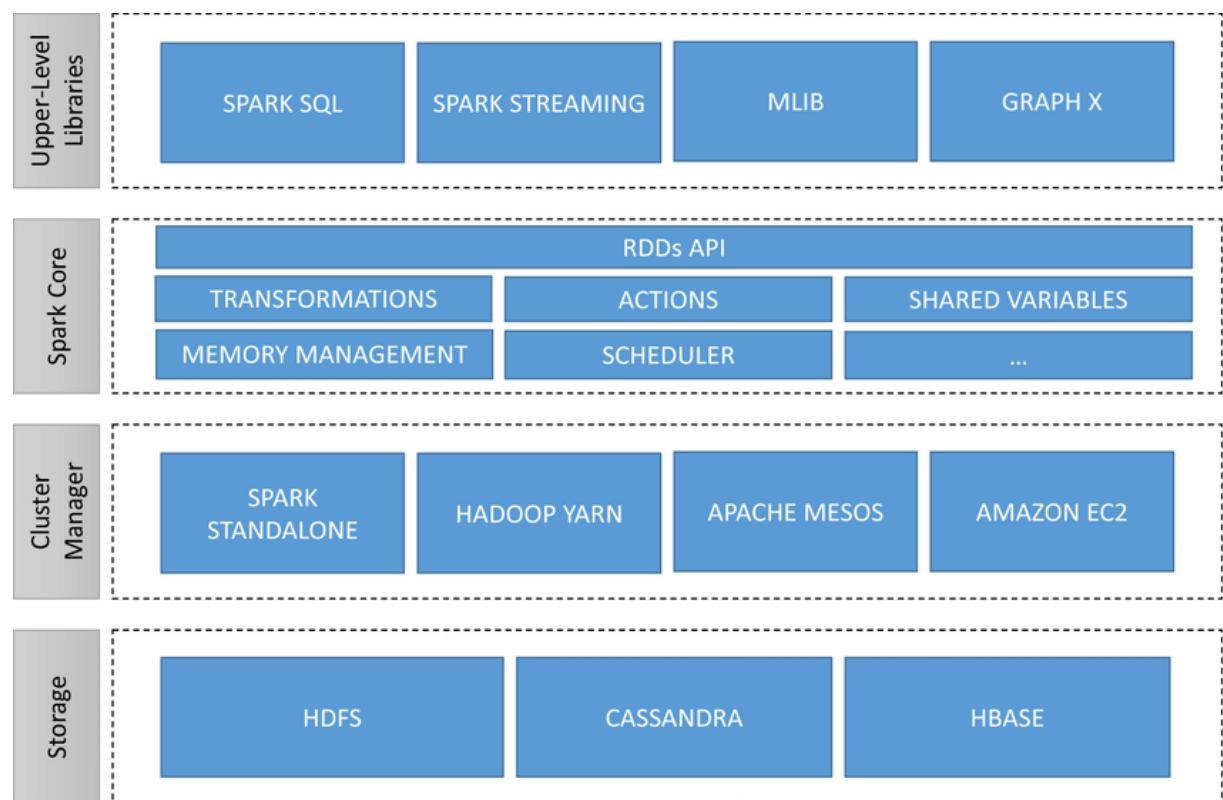
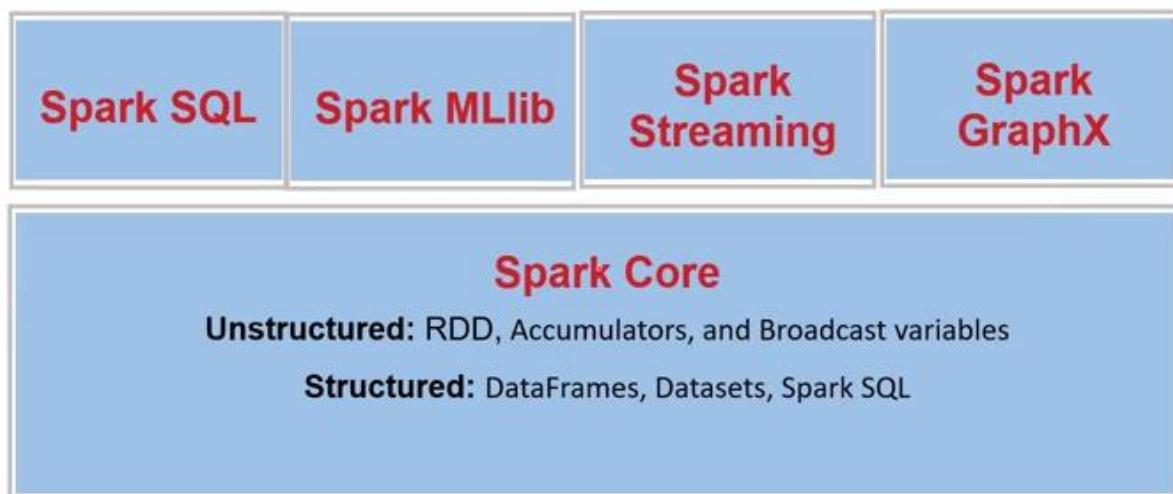
Neden Spark?

- Hızlıdır.
 - Çünkü belleği iyi kullanır.
- Ölçekleme ve Dağıtık Hesaplama yapar.
 - Geniş ölçekli hesaplama yapabilmektedir. Bir bilgisayardan binlerce sunucuya kadar hesaplama yapabilmektedir.
 - Dağıtık veri işleme sayesinde binlerce sunucu üzerinde Petabayt ölçüünde veri işleme yapabilir.
- Arkasında çok güçlü bir topluluk vardır.
 - Databricks
 - Cloudera
 - UC Berkeley

MapReduce ile Farklılıklar

- İkisi de Hadoop YARN üzerinde çalışabilir.
- Spark 100 kata kadar daha hızlı çalışabilir.
- Spark Hadoop'a bağımlı değildir.
- Spark belleği iyi kullanır. Öte yandan MapReduce diske bağımlıdır.
- Spark'ta shuffle daha uygun maliyetlidir.
- Spark'in daha az I/O organizasyonuna ihtiyacı vardır.
- Spark interaktif kullanılabilir (özellikle Makine Öğrenmesinde)
- Erişebilirliği daha yüksektir (Scala, Java, Python, R).
- Geliştiriciler daha verimli çalışabilir.

Spark Stack



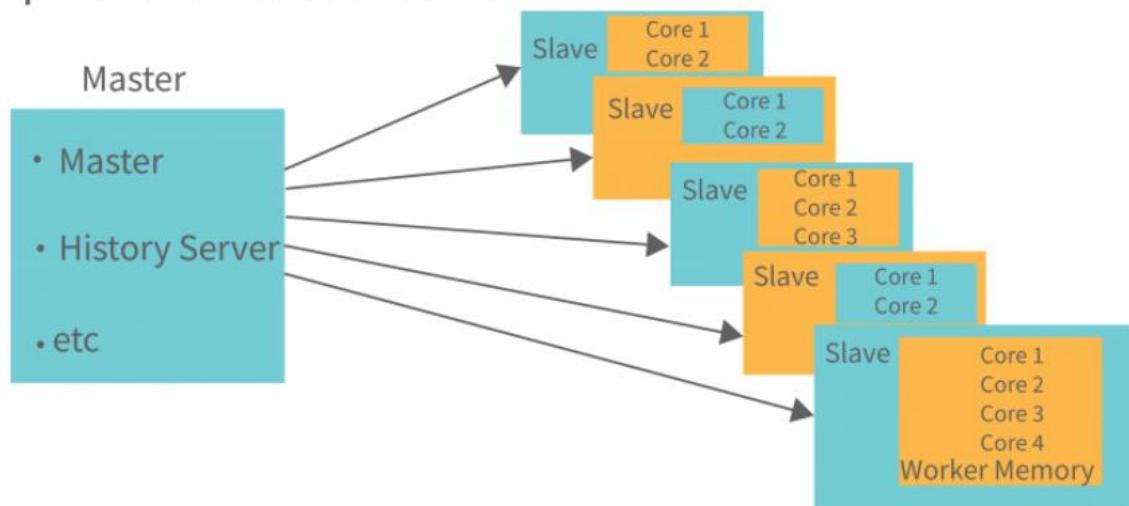
Spark Çalışma Modları

- İki tür çalışma modu vardır.
 - Cluster Mode
 - Production'a yönelik moddur.
 - Cluster'ın da farklı seçenekleri vardır.
 - Spark Standalone Cluster
 - Apache Mesos
 - Hadoop YARN
 - Kubernetes
 - Local Mode
 - Daha çok kendi bilgisayarımızda öğrenmek, geliştirmek ve çalışmak için çalıştırduğumuz, Production için uygun olmayan moddur.

Spark Standalone Cluster

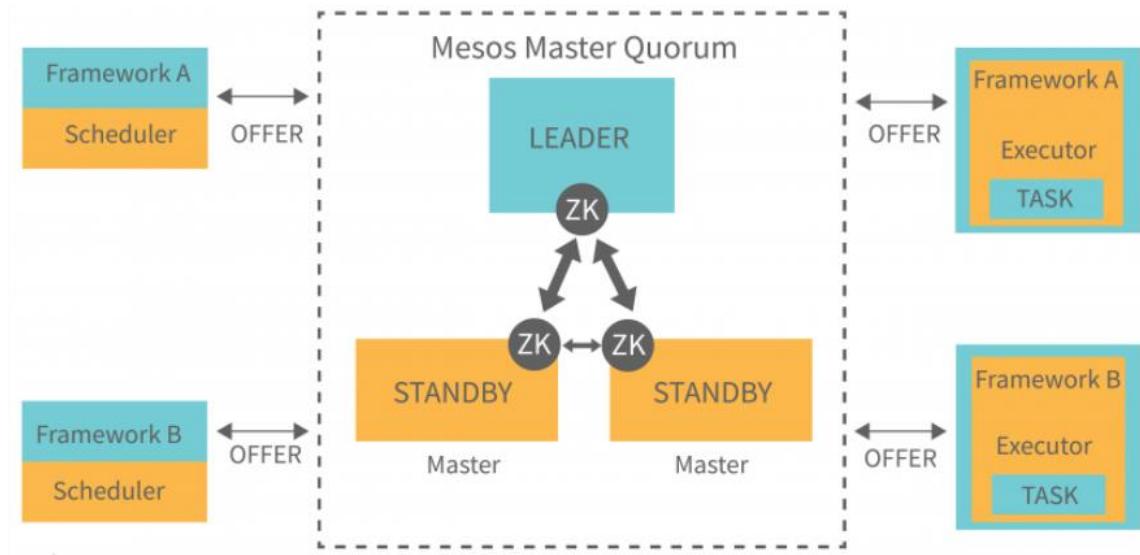
- Cluster kurulumunu kolaylaştırmak için yazılım paketine bir Spark Cluster Manager dahildir.
- Resource Manager ve Worker, bağımsız olan tek Spark Standalone Cluster bileşenleridir.
- Standalone Cluster modunda her worker node'da görevleri çalıştıran yalnızca bir executor vardır.
- Bir istemci Standalone Master ile bağlantı kurduğunda, kaynakları talep ettiğinde ve yürütme sürecini başlattığında, Standalone Clustered yönetici yürütme sürecini başlatır.
- Buradaki müşteri Application Master'dır ve kaynakları Resource Manager'dan ister.

Spark Standalone Architecture



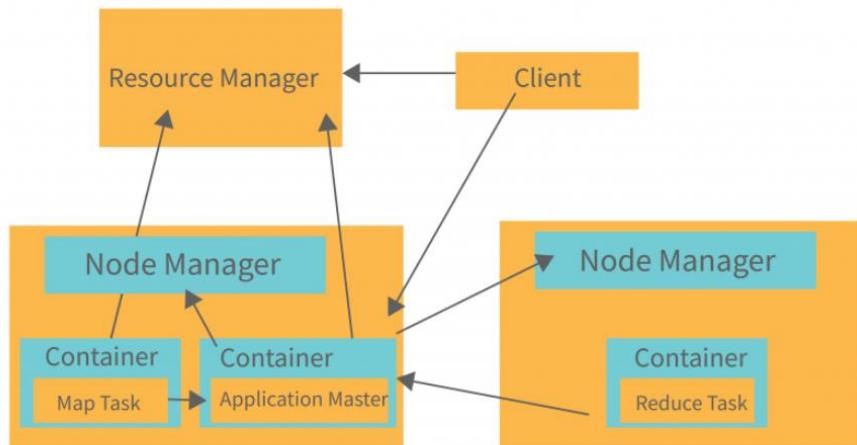
Apache Mesos

- Hadoop MapReduce ve hizmet uygulamalarını çalıştırabildiği gibi genel bir küme yöneticisi de olabilir.
- Apache Mesos, dinamik kaynak paylaşımı ve izolasyonu kullanarak uygulama kümelerinin geliştirilmesine ve yönetimine katkıda bulunur.
- Uygulamaların büyük ölçekli küme ortamlarında konuşlandırılmasına ve yönetilmesine olanak sağlar.
- Mesos çerçevesi üç bileşen içerir:
 - **Mesos Master**
 - Mesos Masterkümesi, hata toleransı (bir arıza meydana geldiğinde çalışma ve kayıptan kurtarma yeteneği) sağlar. Mesos Master tasarıımı nedeniyle, bir küme birçok Mesos Master'i içerir.
 - **Mesos Slave**
 - Mesos Slave, kümeye kaynak sağlayan bir örnektir. Mesos Master bir görev atadığında Mesos Slave kaynak atamaz.
 - **Mesos Framework**
 - Uygulamalar, uygulamanın görevleri gerçekleştirebilmesi için kümeden kaynak talep edebilir. Mesos Framework buna izin verir.



Hadoop Yarn

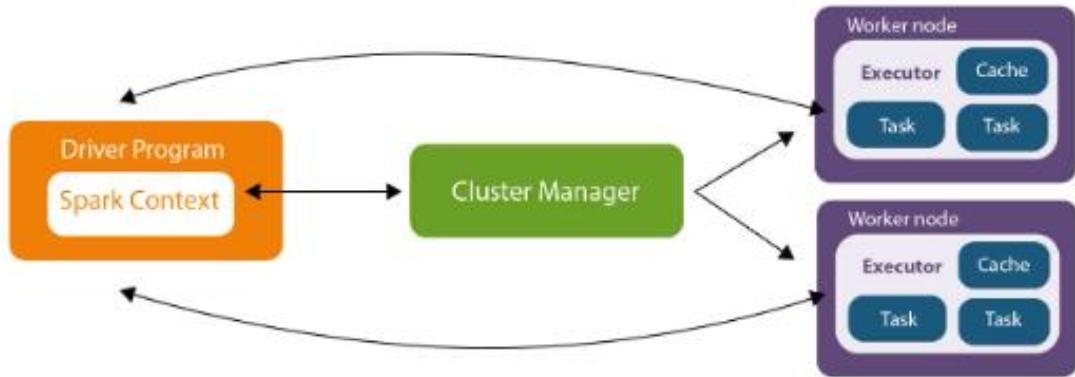
- Hadoop 2.0'in önemli bir özelliği geliştirilmiş resource manager'dır. Hadoop ekosistemi kaynakları yönetmek için YARN'a güveniyor.
- Aşağıdaki iki bileşenden oluşur:
 - **Resource Manager**
 - Sistem kaynaklarının tüm uygulamalara tahsisini kontrol eder. Bir Scheduler ve bir Application Manager dahildir. Uygulamalar kaynakları Scheduler'dan alır.
 - **Node Manager**
 - Her iş veya uygulamanın bir veya daha fazla konteynere ihtiyacı vardır ve Node Manager bu konteynerleri ve bunların kullanımını izler. Node Manager, bir Application Manager ve bir Container Manager'dan oluşur. MapReduce framework her görev bir kapta çalışır. Node Manager kapsayıcıları ve kaynak kullanımını izler ve bu, Resource Manager raporlanır.



Spark Mimarisi

- Driver Program: Çalışacak kod.
- Cluster Manager: Standalone, YARN, Mesos ...
- Worker sunucularda çalışacak olan worker processler (Node Manager)
 - CPU, memory and storage resources
- Worker sunucularda çalışan executor'lar
 - JVM, her uygulamayıń executor'ları ayrıdır, tahsis edildiği kadardır, cache, disk storage

- Worker sunucularda çalışan task'lar
 - Bir executor'da birden fazla thread olabilir.
 - Her thread'de bir task çalışabilir.
 - Hesaplama yapan en temel birimdir.
 - Bir job için çalışacak task'ların sayısı partition sayısına bağlıdır.



Dataframe API Giriş

Structured (Yapısal) Nedir?

- Yapısal: Sütün isimleri ve sakladığı veri tipi belirlidir.
- Yarı Yapısal: Her satırda bir kayıt, ayraç ile ayrılmıştır.

Unstructured data	Semi-structured data	Structured data																								
<p>The university has 5600 students. John's ID is number 1, he is 18 years old and already holds a B.Sc. degree. David's ID is number 2, he is 31 years old and holds a Ph.D. degree. Robert's ID is number 3, he is 51 years old and also holds the same degree as David, a Ph.D. degree.</p>	<pre><University> <Student ID="1"> <Name>John</Name> <Age>18</Age> <Degree>B.Sc.</Degree> </Student> <Student ID="2"> <Name>David</Name> <Age>31</Age> <Degree>Ph.D. </Degree> </Student> ... </University></pre>	<table border="1"> <thead> <tr> <th>ID</th><th>Name</th><th>Age</th><th>Degree</th></tr> </thead> <tbody> <tr> <td>1</td><td>John</td><td>18</td><td>B.Sc.</td></tr> <tr> <td>2</td><td>David</td><td>31</td><td>Ph.D.</td></tr> <tr> <td>3</td><td>Robert</td><td>51</td><td>Ph.D.</td></tr> <tr> <td>4</td><td>Rick</td><td>26</td><td>M.Sc.</td></tr> <tr> <td>5</td><td>Michael</td><td>19</td><td>B.Sc.</td></tr> </tbody> </table>	ID	Name	Age	Degree	1	John	18	B.Sc.	2	David	31	Ph.D.	3	Robert	51	Ph.D.	4	Rick	26	M.Sc.	5	Michael	19	B.Sc.
ID	Name	Age	Degree																							
1	John	18	B.Sc.																							
2	David	31	Ph.D.																							
3	Robert	51	Ph.D.																							
4	Rick	26	M.Sc.																							
5	Michael	19	B.Sc.																							

Spark Structured API

- Üç tür karşımıza daha çok çıkmaktadır.
 - Dataframe
 - Dataset
 - SparkSQL

Schema

- Sütun isimleri ve veri tiplerinin daha önceden bilinmesidir.

- Spark ne ile karşılaşacağının bildiği için daha rahattır.
- Yapısal olmayan API ile en büyük fark budur.
- Kullanıcı isterse kendisi bir şema tanımlayabilir.
- Kullanıcı, veriyi yüklerken bir çıkarımda bulunabilir.

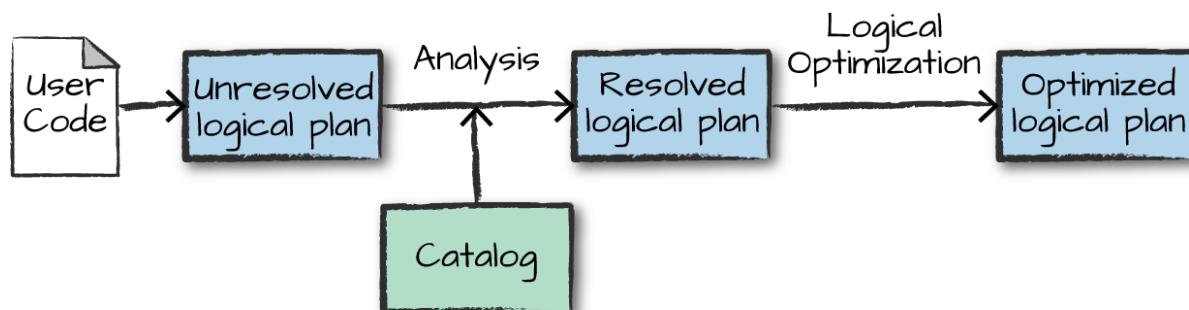
Spark Dataframe

- Tabloya benzetebiliriz.
- Sütun isimleri ve türlerine, şemaya sahiptir.
- R ve Pandas Dataframe'ne benzerdir. Tek farkı dağıtık olmasıdır.
- R ve Pandas Dataframe'leri Spark Dataframe'e dönüştürüyor.
- Kullanımı kolay ve tavsiye ediliyor (RDD (Resilient Distributed Dataset), Dataset, SQLTable)

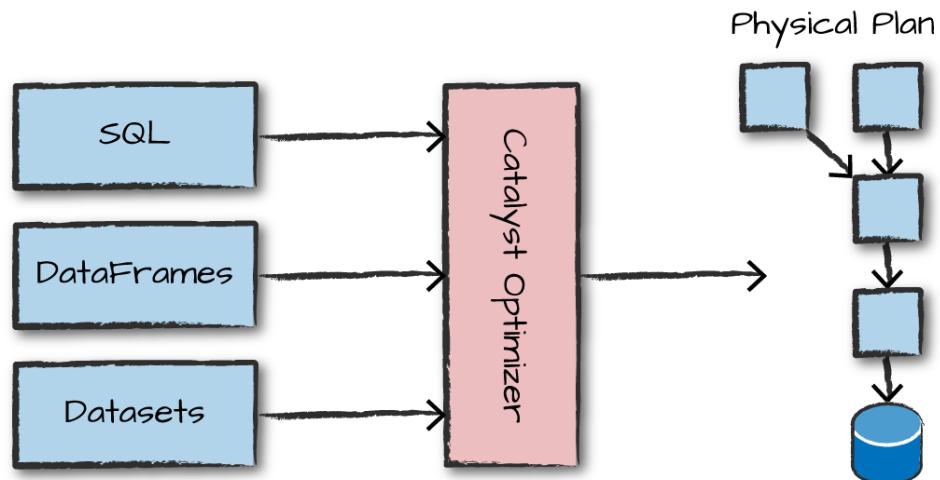
Spark Datasets

- Java ve Scala gibi derlenebilen dillere özgündür.
- Dataframe'e ek olarak kullanıcı veri tip tanımlayabiliyor (case class).
- Dataframe kullanımı daya yayındır.

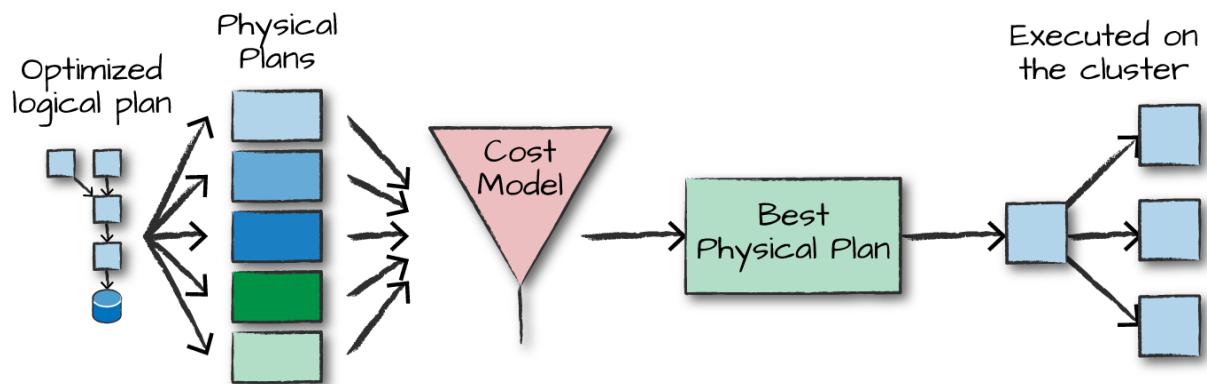
Spark Structured API Çalışma Planı



- SQL/DataFrame/Dataset kodu yaz.
- Eğer kod geçerli ise Spark mantıksal plan oluşturur.
- Spark, bu Mantıksal Planı Fiziksel Plana dönüştürerek yol boyunca optimizasyonları kontrol eder.
- Spark daha sonra bu Fiziksel Planı (RDD manipülasyonları) cluster üzerinde yürütür.



Physical Planning

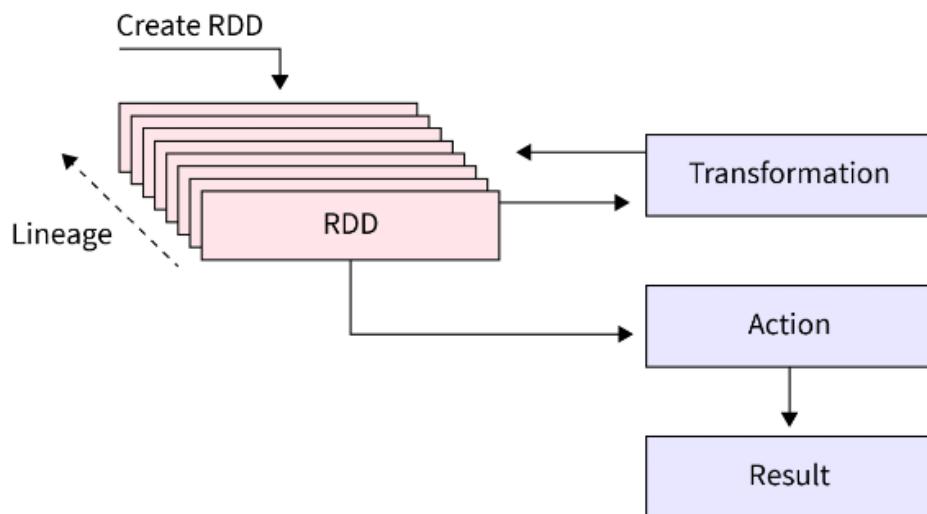


- Optimize edilmiş bir mantıksal planı başarıyla oluşturduktan sonra Spark, fiziksel planlama sürecine başlar.
- Genellikle Spark planı olarak adlandırılan fiziksel plan farklı fiziksel yürütme stratejileri üreterek ve bunları bir maliyet modeliyle karşılaştırarak mantıksal planın küme üzerinde nasıl yürütüleceğini belirtir.
- Maliyet karşılaştırmasına bir örnek, belirli bir tablonun fiziksel özelliklerine (tablonun boyutu veya bölümlerinin boyutu) bakarak belirli bir birleşimin nasıl gerçekleştirileceğini seçmek olabilir.
- Fiziksel planlama bir dizi RDD ve dönüşümle sonuçlanır. Spark'in derleyici olarak adlandırılmasının nedeni bu sonuktur; DataFrames, Datasets ve SQL'deki sorguları alır ve bunları sizin için RDD dönüşümlerine derler.

Lazy Evaluation, Transformations and Action

Lazy Evaluation

- Lazy Evaluation, Apache Spark'ta veriler üzerindeki dönüşümlerin hemen yürütülmediği, bunun yerine yürütmenin bir action tetiklenene kadar ertelendiği önemli bir kavramdır.
- Spark'ta transformations, bir giriş RDD'sini (Esnek Dağıtılmış Veri Kümesi) veya DataFrame'i yeni bir RDD'ye veya DataFrame'e dönüştüren işlemlerdir.
- Spark'ta transformations çağrıldığında sonuçları hemen hesaplanmaz. Bunun yerine Spark, uygulanacak dönüşümlerin sırasını temsil eden DAG (Yönlendirilmiş Döngüsel Olmayan Grafik) adı verilen mantıksal bir yürütme planı oluşturur. DAG, hesaplamanın dahili bir temsilidir.
- Spark'taki Lazy Evaluation, transformations hiçbir zaman yürütülmeyeceği anlamına gelmediğini unutmamak önemlidir. Dönüşümlerin yürütülmesi, RDD veya DataFrame'de bir action çağrılmaya kadar ertelenir. Spark'taki action tüm DAG'in yürütülmesini tetikler ve hesaplanan sonuçlar sürücü programına döndürülür veya harici bir depolama sistemine yazılır.



Transformations:

- Apache Spark'taki dönüşümler, yeni RDD'ler veya DataFrames oluşturmak için dağıtılmış veri kümelerine (RDD'ler) veya yapılandırılmış veri kümelerine (DataFrames veya Datasets) uygulanan işlemlerdir.

- Dönüşümler tembel bir şekilde değerlendirilir, yani hemen yürütülmeler, bunun yerine uygulanacak dönüşüm sırasını temsil eden mantıksal bir yürütme planı (DAG) oluştururlar.
- Bir eylem tetiklendiğinde dönüşümler optimize edilmiş bir şekilde yürütülür.
- Spark'taki dönüşümler veri manipülasyonunu, filtrelemeyi, toplamaları ve daha fazlasını mümkün kılar.
- Dönüşümler değiştirilemez; yani orijinal RDD'yi veya DataFrame'i değiştirmezler ancak uygulanan dönüşümle yeni bir tane oluştururlar.
- İki tür transformation vardır:
 - Narrow Transformation (map, flatmap, filter, union ...)
 - Wide Transformation (groupBy, join, aggregateByKey ...)
- Spark'ta yer alan bazı yaygın transformation'lar:

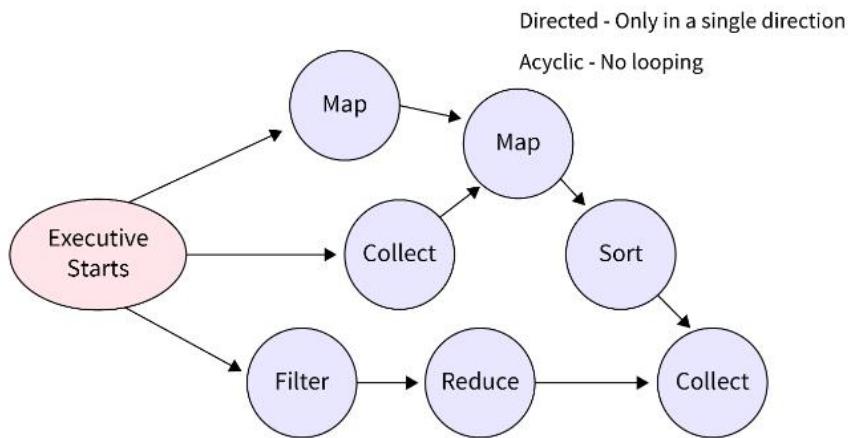
◦ Map	◦ Sort
◦ Filter	◦ Sample
◦ FlatMap	◦ Union
◦ GroupBy	◦ Join
◦ ReduceByKey	◦ Aggregations
◦ Distinct	

Actions:

- Apache Spark'taki eylemler, hesaplamaların yürütülmesini tetikleyen ve sonuçları döndüren veya harici bir sisteme veri yazan işlemlerdir. Tembelce değerlendirilen dönüşümlerin aksine, eylemler DAG'in yürütülmesini tetikler, bu da dönüşümlerin değerlendirilmesine ve nihai sonuçların üretilmesine yol açar.
- Eylemler istekli işlemlerdir ve özellikle büyük veri kümeleriyle çalışırken önemli hesaplama ve G/C maliyetlerine neden olabilir.
- Spark'ta yaygın olarak kullanılan action'lar:

◦ Collect	◦ Aggregate
◦ Count	◦ Save
◦ First	◦ Foreach
◦ Take	◦ CountByKey
◦ Reduce	◦ CollectAsMap

DAG (Directed Acyclic Graph)



Lazy Evaluation'in Avantajları:

- Optimize Edilmiş Performans
- Kaynak Verimliliği
- Esneklik ve Kompozisyon
- Dinamik Kontrol Akışı
- Hata Kontrol Etme ve Hata Toleransı

Spark Optimizasyon Teknikleri

- Kaynakların doğru şekilde kullanılmasını ve işlerin hızlı bir şekilde yürütülmesini sağlamak amacıyla Spark'ın ayarlarını ve özelliklerini değiştirmek için Spark optimizasyon teknikleri kullanılır.

Data Serialization

- Burada bellek içi nesne, bir dosyada saklanabilecek veya ağ üzerinden gönderilebilecek başka bir formata dönüştürülür. Bu, dağıtılmış uygulamaların performansını artırır.
- En iyi Spark optimizasyon tekniğidir. Serileştirme, herhangi bir dağıtılmış uygulamanın performansını artırır.
- Spark varsayılan olarak JVM platformu üzerinden Java serileştiricisini kullanır. Spark ayrıca Java serileştiricisi yerine Kryo olarak bilinen bir serileştiriciyi de kullanabilir. Kryo seri hale getirici, Java seri hale getiriciden daha iyi performans sağlar.

Cache and Persist

- Bu, verilere daha sık ihtiyaç duyulduğunda kullanılan etkili bir tekniktir. Cache() ve persist() bu teknikte kullanılan yöntemlerdir.

- Bu yöntemler bir RDD, DataSet ve DataFrame'in hesaplamalarını depolamak için kullanılır. Ancak, cache() onu bellekte saklar ve persist() onu kullanıcı tanımlı depolama seviyesinde saklar.
- Bu yöntemler, tekrarlanan hesaplamalar kullanıldığından maliyetlerin azaltılmasına ve zamandan tasarruf edilmesine yardımcı olabilir.
- Cache ve Persist yöntemleri ihtiyaç olduğunda veri setini hafızaya kaydedecektr. Programınızda sıkılıkla kullanılan küçük bir veri kümesini depolamak istediğinizde kullanışlıdır.

Data Structure Tuning

- Spark'ı kullanırken, ek yük yaratabilecek bazı Java özelliklerini değiştirek bellek tüketimini azaltabiliriz. Bu aşağıdaki şekillerde mümkündür:
 - Anahtarlar için dizeler yerine numaralandırılmış nesneler veya sayısal kimlikler kullanın.
 - Avoid using a lot of objects and complicated nested structures.
 - Bellek boyutu 32 GB'tan azsa JVM bayrağını xx:+UseCompressedOops olarak ayarlayın

Garbage Collection Optimizasyonu

- Garbage collectors optimize etmek için Spark uygulamalarını çalıştırırmak üzere G1 ve GC kullanılmalıdır.
- G1 toplayıcı büyüyen yığınları yönetir. Uygulamaların beklenmeyen davranışlarını kontrol etmek için oluşturulan günlüklere göre GC ayarı önemlidir. Ancak bundan önce programın mantığını ve kodunu değiştirip optimize etmeniz gerekiyor.
- G1GC, işlemler arasındaki duraklama sürelerini optimize ederek işlerin yürütme süresinin azaltılmasına yardımcı olur.

API Seçimi

- Spark, üzerinde çalışılacak üç tür API'yi tanıttı: RDD, DataFrame, DataSet
- RDD, daha az optimizasyonla düşük seviyeli işlemler için kullanılır
- DataFrame, katalizör optimize edicisi ve düşük çöp toplama (GC) yükü nedeniyle çoğu durumda en iyi seçimidir.
- Veri kümesi son derece güvenlidir ve kodlayıcılar kullanır. İkili formatta serileştirme için Tungsten kullanır.

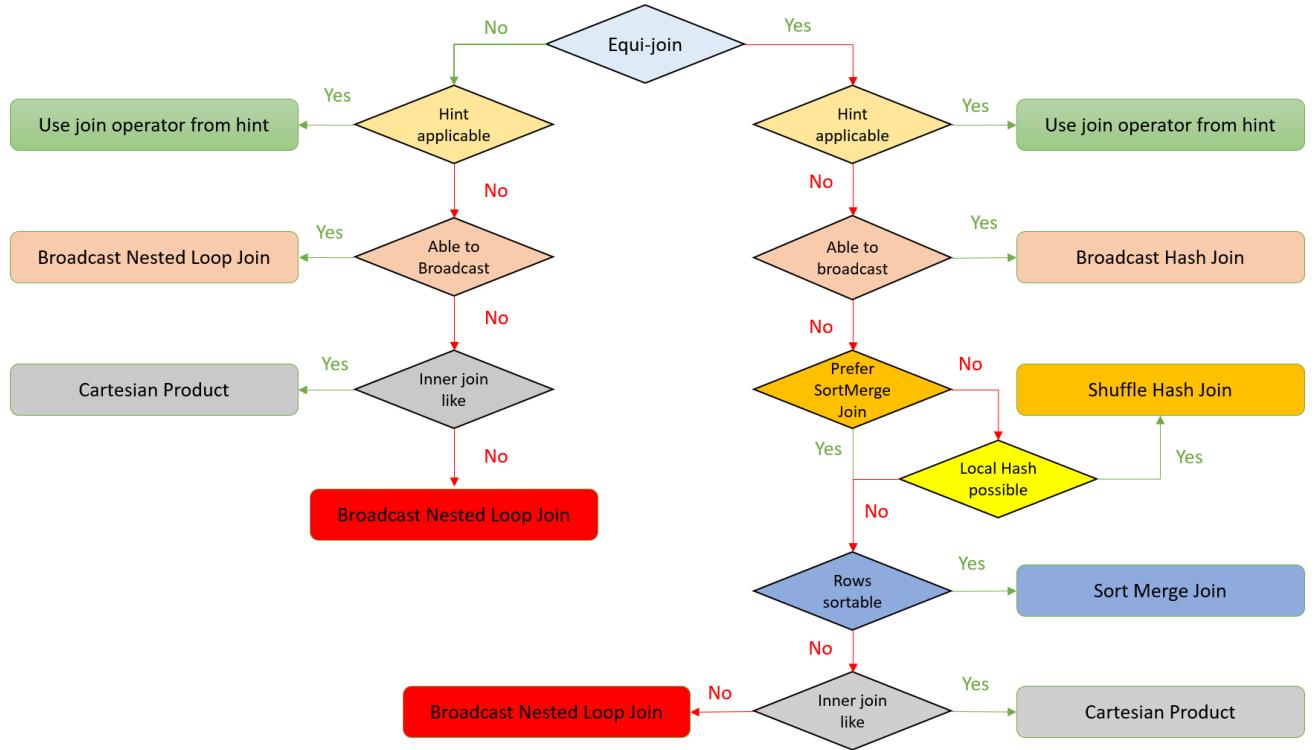
UDF Kullanmaktan Kaçınmak

- Kullanıcı Tanımlı İşlevler (UDF'ler), kullanıcı tarafından tanımlanan işlevlerdir ve Spark'in SQL ve DataFrame API'lerinin işlevsellliğini genişletmek için kullanılabilir. UDF'ler bazı durumlarda yararlı olabilse de PySpark uygulamalarının performansı üzerinde de olumsuz etkiler yaratabilir.
- UDF'lerin performansı etkileyebilmesinin ana nedenlerinden biri, verilerin dağıtıldığı Worker Nodes yerine Spark uygulamasının Driver Node'da yürütülmeleridir. Bu, özellikle büyük veri kümeleri ve karmaşık UDF'ler için iletişim yükünün artmasına ve yürütme sürelerinin yavaşlamasına neden olabilir. Ayrıca UDF'ler, DataFrame/Dataset API'nin tüm performans avantajlarını ortadan kaldırır.
- UDF'lerin performans üzerindeki olumsuz etkisini önlemek için genellikle mümkün olduğunda Spark'in yerleşik işlevlerinin ve API'lerinin kullanılması önerilir.

Shuffle Kullanmaktan Kaçının

- Shuffle, çok fazla bellek tüketen ağır bir işlemidir.
- Spark'ta kodlama yaparken kullanıcı her zaman herhangi bir shuffle işleminden kaçınmaya çalışmalıdır çünkü karıştırma işlemi performansı düşürecektir. Yüksek shuffle varsa, kullanıcı out of memory hatası alabilir. Bu durumda bu hatayı önlemek için kullanıcının paralellik düzeyini artırması gereklidir.
- GroupBy yerine kullanıcı, azaltıcıByKey'i tercih etmelidir çünkü groupByKey, performansı engelleyen çok fazla karıştırma oluştururken, azaltıcıByKey verileri o kadar fazla karıştırmaz. Bu nedenle, azaltByKey, groupByKey ile karşılaştırıldığında daha hızlıdır. Herhangi bir ByKey işlemi kullanıldığında kullanıcı verileri doğru bir şekilde böülümlendirmelidir.

Spark Join Strategies



- Verilerle uğraşırken Join kaçınılmazdır ve sorgu yazarken hepimiz en az bir kez inner, outer, left, right veya semi-join gibi sorguları yazarken Join'ler ile karşılaşmışızdır.
- Ancak Apache Spark söz konusu olduğunda bu basit birleştirmeler çok farklı şekilde ele alınır. Apache Spark, birleştirme işlemini gerçekleştirmek için 5 farklı strateji kullanır.

Bunlar:

- Broadcast Hash Join (BHQ)
- Shuffle Hash Join (SHQ)
- Sort Merge Join (SMQ)
- Broadcast Nested Loop Join (BNLQ)
- Cartesian Product Join (Shuffle-and-Replication Nested Loop Join) (CPQ)

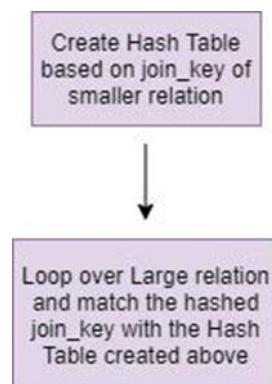
Join'lerde Etkili Olan Faktörler

- Veri Boyutu
 - Birleştirmeye katılan veri kümelerinin boyutu, birleştirme işleminin performansını doğrudan etkileyecektir.
- Veri Dağılımı
 - Birleştirilen veriler küme genelinde eşit şekilde dağılmazsa veri çarpıklığına (skew) neden olabilir ve bu da performans sorununa neden olabilir.
- Birleştirme Tipi ve Birleştirme Koşulu

- Inner Join, Outer Join ve Left Join gibi farklı birleştirme türleri, birleştirilen verilere bağlı olarak değişen performans özelliklerine sahip olabilir. Örn: equi-join — “=” / non-equi-join — “<, >, \geq , \leq ”
- Veri Formatı
 - Birleştirilen verilerin formatı da performansı etkileyebilir. Örneğin Parquet veya ORC formatı, columnar depolama ve sıkıştırma yetenekleri nedeniyle CSV veya JSON formatından daha verimli olabilir.

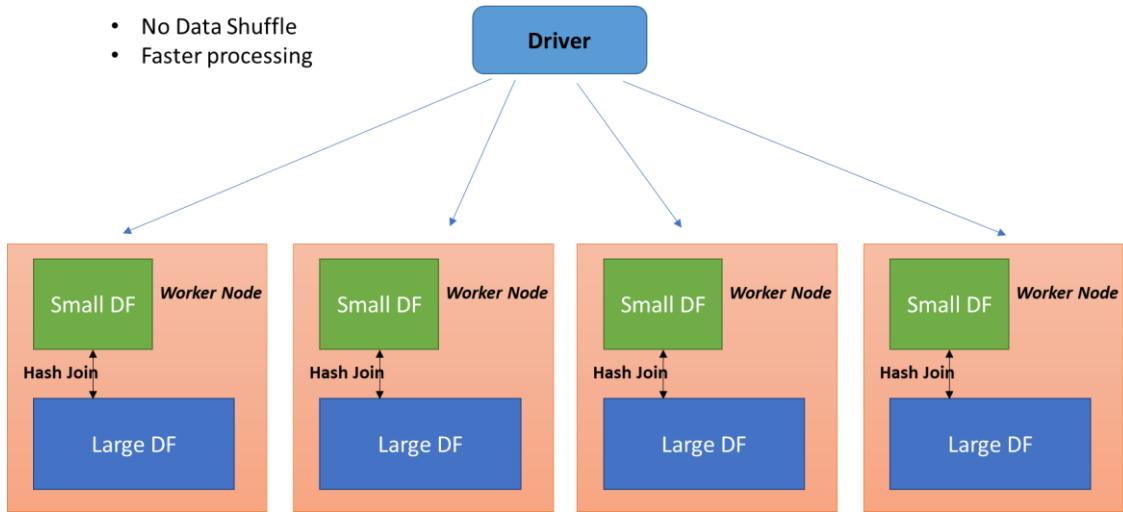
Hash Joins Nedir?

- Hash Joins, daha küçük veri kümelerinin birleştirme anahtarına dayalı olarak bir hash tablosu oluşturulur ve ardından Hash Join anahtarı alanlarıyla eşleşmek için daha büyük veri kümeleri üzerinde döngü yapılır. Yalnızca denklik birleştirme koşulunu destekler.
- Bu strateji, düğüm başına düzeyde (veri kümelerinin mevcut olduğu düğümlerdeki tüm bölgüler) uygulanır.



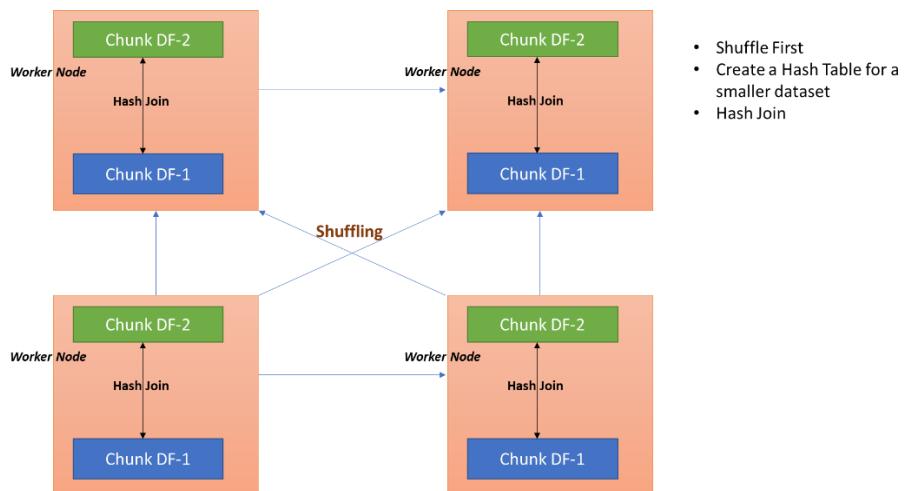
Broadcast Hash Joining

- Broadcast Hash join, daha küçük veri kümelerini çalışan düğümlere yayılmamak için basit bir strateji kullanır ve böylece Shuffle maliyetinden tasarruf sağlar.
- Bu tür birleştirme, veri kümelerinden **biri çok büyük** ve diğer genellikle veri belleğe tamamen sığacak kadar **küçük** olduğunda (varsayılan **10 MB**'tan) oldukça kullanışlıdır. Bu birleştirmeye **Map End Join** de denir. Başlangıçta bu tablo Driver Node'da depolanır ve ardından tüm Executor Node'lara yayılır.
- Sadece “=” join (equi-join) için uygundur.
- Full Outer Join dışında bütün tipleri (inner, left, right) destekler.



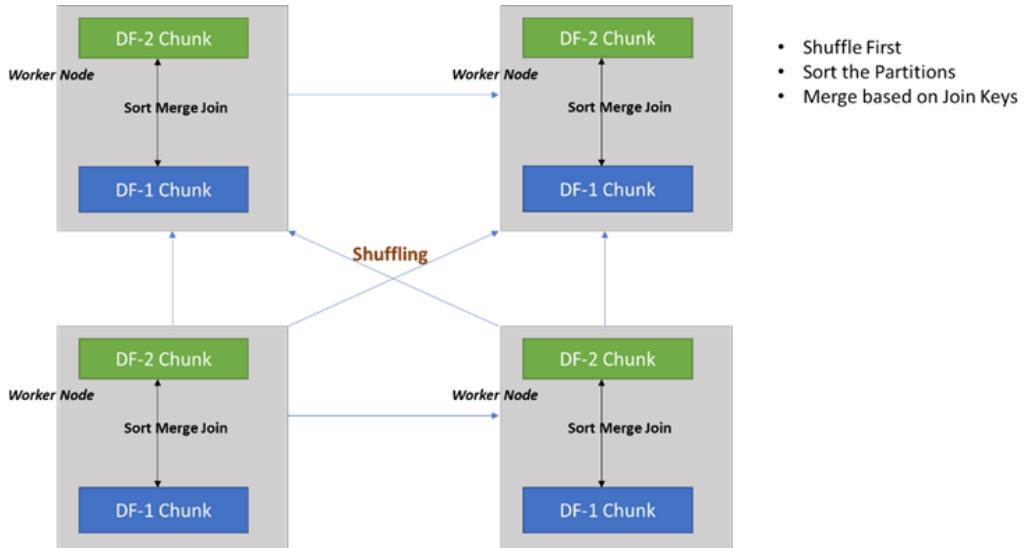
Shuffle Hash Joining

- Her iki tablo da büyük olduğunda yayının birleştirilmesi, yetersiz bellek sorunlarına yol açabilir. Bu durumda Shuffle Hash join daha iyi bir seçenek olabilir.
- Shuffle Hash Join, shuffle ve hash birleştirme aşaması olmak üzere iki aşamalı bir işlemi içerir.
- Shuffle Hash Join, aynı executor node aynı birleştirme anahtarı değerine sahip verilerin ve ardından Hash Join taşınmasını içerir.
- Sadece “=” join (equi-join) için uygundur.



Shuffle Sort Margin Join

- Shuffle Sort Merge Join adından da anlaşılacağı gibi shuffle, sort-merge aşamasını içerir. Aynı birleştirme anahtarı sahip veri kümeleri aynı Executor Node taşınır ve ardından Executor Node, düğümdeki veri kümlesi bölümleri birleştirme anahtarlarına göre sıralanır ve ardından birleştirme anahtarlarına göre birleştirilir.
- Sadece “=” join (equi-join) için uygundur.



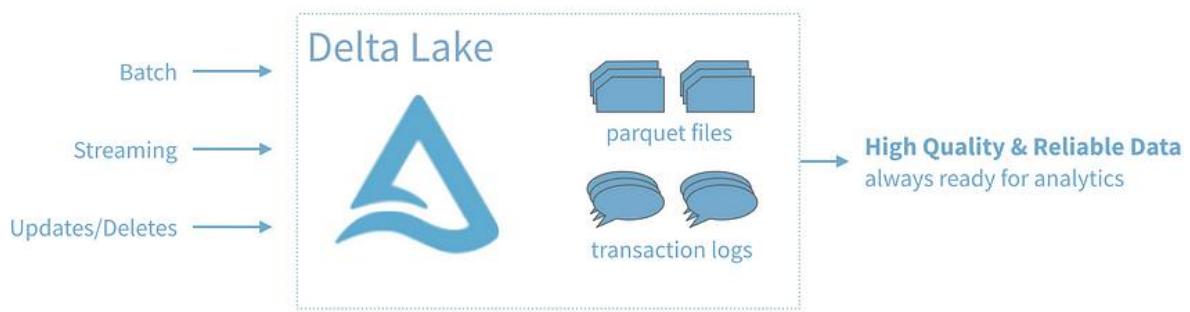
Cartesian Product Join

- Cartesian Product Join (CPJ), Spark'ta **her iki veri kümelerindeki kayıtların tüm olası kombinasyonlarını oluşturarak** iki veri kümelerini birleştirmek için kullanılan bir yöntemdir.
- CPJ'de, bir veri kümelerindeki her kayıt diğer veri kümelerindeki her kayıtla eşleştirilir ve iki veri kümelerinin Cartesian Product elde edilir.
- Cartesian Product Join, kayıtların tüm olası kombinasyonlarını oluşturduğundan, özellikle büyük veri kümeleriyle uğraşırken çok masraflı bir işlem olabilir.

Broadcast Nested Loop Join (BNLP)

- Broadcast Nested Loop Join, veri kümelerinden birinin tek bir düğümün belleğine sığacak kadar küçük olduğu iki veri kümelerini birleştirmek için Spark'ta kullanılan bir yöntemdir. Bu yöntemde, daha küçük veri kümeleri kümelerdeki tüm düğümlere yayınlanır ve ardından daha büyük veri kümeleriyle iç içe döngü birleştirme işlemi gerçekleştirilir.

Delta Lake



- Delta Lake, Databricks lakehouse veri ve tabloların depolanması için temel sağlayan optimize edilmiş depolama katmanıdır.
- Delta Lake, Parquet veri dosyalarını **ACID** işlemleri ve ölçeklenebilir meta veri işleme için dosya tabanlı bir işlem günlüğüyle genişleten açık kaynaklı bir yazılımdır.
- Delta Lake, Apache Spark API'leriyle tamamen uyumludur ve Structured Streaming ile sıkı entegrasyon için geliştirildi. Hem toplu hem de akış işlemleri için tek bir veri kopyasını kolayca kullanmanıza olanak tanır ve uygun ölçüde artımlı işleme sağlar.
- Delta Lake'in sunmuş olduğu özellikler:
 - ACID Transactions
 - Delta Lake, ACID işlemlerini data lakes'e getirir.
 - Serileştirilebilir izolasyon seviyeleri, okuyucuların asla tutarsız veriler görememesini sağlar.
 - Atomicity, Consistency, Isolation and Durability.
 - Scalable Metadata Handling
 - Milyarlarca dosya içeren petabayt ölçekli tablolara yönelik tüm meta verileri kolaylıkla işlemek için Spark'in dağıtılmış işlem gücünden yararlanır.
 - Streaming and batch unification
 - Delta Lake'teki bir tablo, bir toplu tablonun yanı sıra bir akış kaynağı ve havuzudur. Akışlı veri alımı, toplu geçmiş dolgusu, etkileşimli sorguların tümü kutudan çıktıgı gibi çalışır.
 - Time Travel (data versioning)
 - Veri sürümü oluşturma, geri alma işlemlerine, tam geçmiş denetim yollarına ve tekrarlanabilir makine öğrenimi deneylerine olanak tanır.
 - Schema Enforcement

- Besleme sırasında hatalı kayıtların eklenmesini önlemek için şema varyasyonlarını otomatik olarak yönetir.
- Upserts and deletes
 - Veri yakalama, yavaş değişen boyut (SCD) işlemleri, akış yükseltmeleri vb. gibi karmaşık kullanım durumlarını etkinleştirmek için birleştirme, güncelleme ve silme işlemlerini destekler.

Spark Machine Learning

Spark ML Pipelines

- Pipelines, ML iş akışlarında standardizasyonu ve kolaylığı sağlar.
- Scikit-learn'den esinlenilmiştir.
- ML Pipelines kullanmanın en büyük faydası hiper parametre optimizasyonudur.

Spark ML Temel Kavramlar

- **Dataframe:** İçinde text, feature vectors, true labels ve prediction ile ilgili bilgiler olabilir.
- **Transformer:** Bir DF'i başka bir DF'e dönüştürür. Örneğin ML Model
- **Estimator:** Kendini uyduracak bir DF arar. Sonra onu Transformer'a dönüştürür.
- **Pipeline:** ML akışında Transformer ve Estimator'ları zincirleme birbirine bağlar.

Real-Time Data Processing with Spark

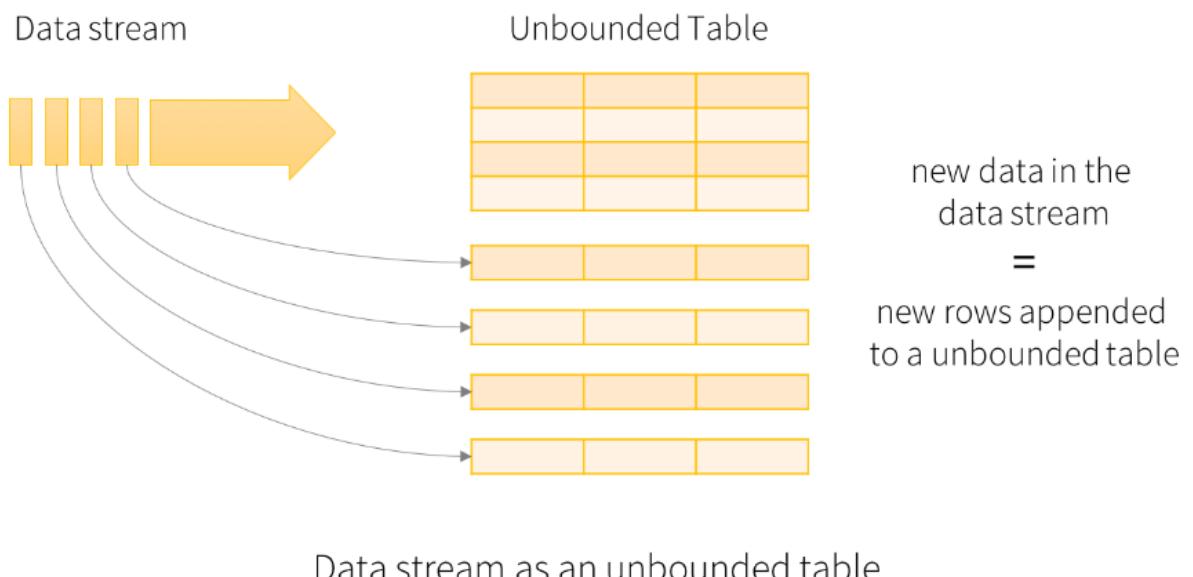
Spark Structured Streaming Giriş

- Spark SQL motoru üzerine inşa edilmiştir.
- Ölçeklenebilir
- Hataya dayanıklı
- Hızlı
- End-to-end-exactly-one
- Akan veri işleme motorudur.

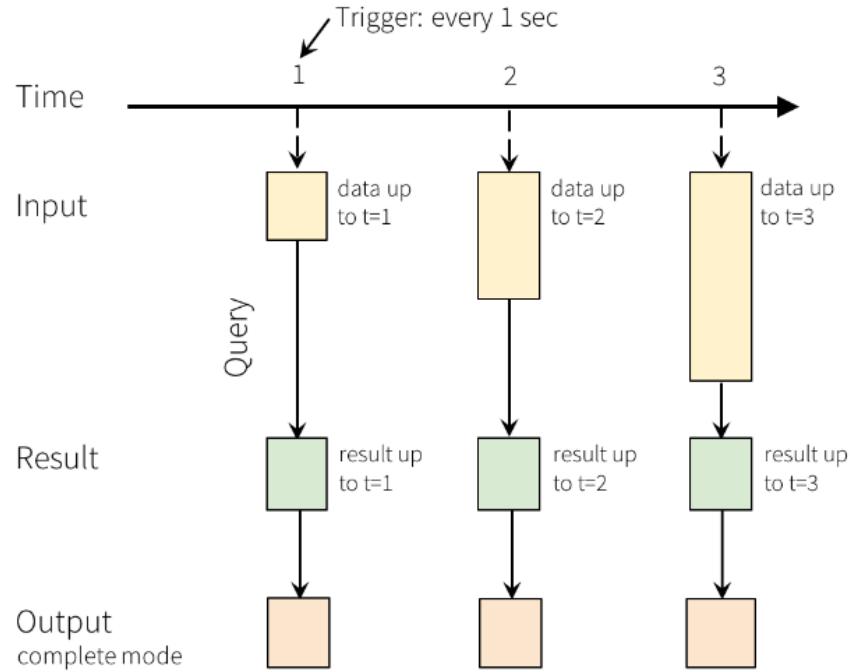
Spark Structured Streaming Programming Model

Basic Concept

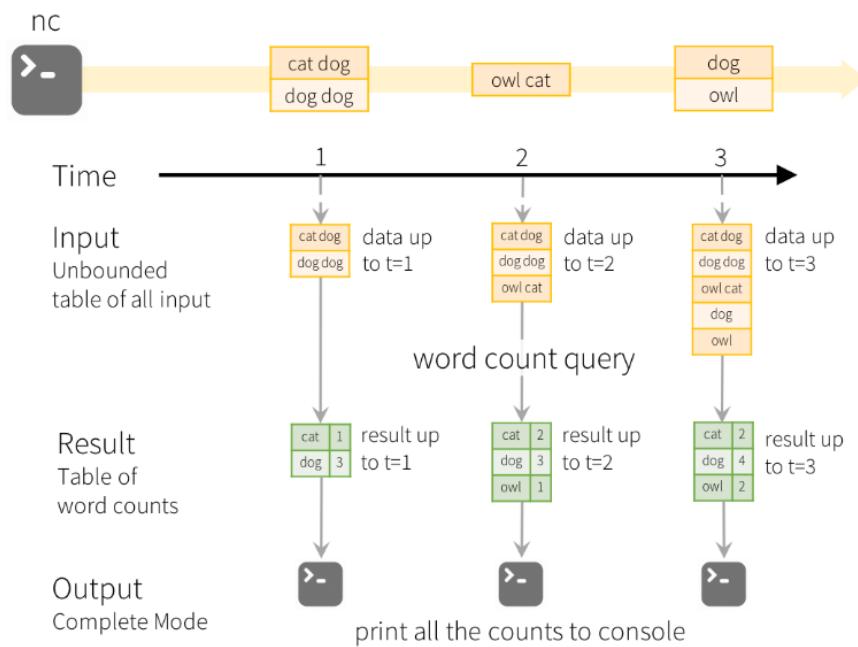
- Spark Structured Streaming, canlı veri akışlarını mikro grplara bölgerek ve sanki statik bir tablodaki toplu sorgularmış gibi işleyerek yönetir.
- Ortaya çıkan çıktı, yeni veriler geldikçe sürekli olarak güncellenir ve verilere ilişkin gerçek zamanlı bilgiler sağlanır. Bu akışlı veri işleme modeli, batch processing 'e benzer.
- Spark Structured Streaming input veri akışı, Spark'in DataFrame API'si kullanılarak sorgulanabilen unbounded (sınırsız) bir tablo olarak değerlendirilir. Her bir mikro veri kümesi, sınırlanmamış tablodaki yeni bir satır "chunk"ı olarak ele alınır ve sorgu motoru, tıpkı normal bir toplu sorgu gibi, unbound tabloya işlemler uygulayarak bir sonuç tablosu oluşturabilir. Sonuç tablosu, yeni veriler geldikçe sürekli olarak güncellenir ve akış verilerinin gerçek zamanlı bir görünümünü sağlar.



- Spark, verileri trigger'lar tarafından tanımlanabilen mikro gruplar halinde işleyecektir. Örneğin bir trigger 1 saniye olarak tanımladığımızı varsayıyalım, bu Spark'in her saniye mikro partiler oluşturup bunları buna göre işleyeceği anlamına geliyor.



Programming Model for Structured Streaming



Model of the Quick Example

Handling Event-time and Late Data

- **Event-time**, verinin kendisinde gömülü olan zamandır. Birçok uygulama için bu olay zamanında işlem yapmak isteyebilirsiniz. Örneğin, IoT cihazları tarafından her dakika oluşturulan olayların sayısını almak istiyorsanız, Spark'in aldığı zaman yerine muhtemelen verilerin oluşturulduğu zamanı (yani verilerdeki olay zamanını) kullanmak istersiniz.
- Bu olay zamanı bu modelde çok doğal bir şekilde ifade edilir; cihazlardan gelen her olay tabloda bir satırdır ve olay zamanı da satırda bir sütun değeridir. Bu, window-based aggregations (örneğin, her dakikadaki olay sayısı) olay zamanı sütununda yalnızca özel bir gruptama ve toplama türü olmasına olanak tanır; her zaman penceresi bir gruptur ve her satır birden fazla pencereye/gruba ait olabilir. Bu nedenle, bu tür olay-zaman penceresi tabanlı toplama sorguları, hem statik bir veri kümesinde (örneğin, toplanan cihaz olay günlüklerinden) hem de bir veri akışında tutarlı bir şekilde tanımlanabilir ve bu da kullanıcının hayatını çok daha kolaylaştırır.
- Ayrıca bu model, doğal olarak beklenenden daha geç gelen verileri olay zamanına göre işler. Spark, Result Table güncellendiğinden, **Late Data** olduğunda eski toplamların güncellenmesi ve ayrıca ara durum verilerinin boyutunu sınırlamak için eski toplamların temizlenmesi üzerinde tam kontrole sahiptir. Spark 2.1'den bu yana, kullanıcının geç veri eşğini belirlemesine ve motorun buna göre eski durumu temizlemesine olanak tanıyan **watermark** desteğimiz var. Bunlar daha sonra **Window Operations** bölümünde daha ayrıntılı olarak açıklanacaktır.

Fault Tolerance

- Streaming işleme uygulamaları, sistemdeki bazı arızalardan sonra mesajların yeniden işlenmesini nasıl ele aldıkları konusunda farklı yaklaşımlar benimser:
 - **At least once**: Her mesajın işlenmesi garanti edilir ancak birden fazla kez işlenebilir.
 - **At most once**: Her mesaj işlenebilir veya işlenmeyebilir. Bir mesaj işlenirse yalnızca bir kez işlenir.
 - **Exactly once**: Her mesajın bir kez ve yalnızca bir kez işlenmesi garanti edilir.
- Structured Streaming'in tasarımının temel hedeflerinden biri end-to-end exactly-once semantics sunmaktadır.
- Bunu başarmak için, Structured Streaming kaynaklarını, sink ve execution engine, işlemin tam ilerlemesini güvenilir bir şekilde izleyecek ve böylece yeniden başlatma

ve/veya yeniden işleme yoluyla her türlü arızanın üstesinden gelebilecek şekilde tasarlandı.

- Her akış kaynağının, akıştaki okuma konumunu izlemek için uzaklıklara sahip olduğu varsayıılır.
- Motor, her trigger'da işlenen verilerin ofset aralığını kaydetmek (checkpointing) için kontrol noktası oluşturma ve yazma öncesi günlükleri kullanır.
- Streaming Sinks, yeniden işlemeyi gerçekleştirmek için bağımsız olacak şekilde tasarlanmıştır. Yeniden oynatılabilir kaynakları ve bağımsız sink'ler birlikte kullanan Structured Stream, herhangi bir arıza durumunda end-to-end exactly-once semantik sağlayabilir.

[API using Datasets and DataFrames](#)

- Spark 2.0'dan bu yana, DataFrames ve Datasets statik, sınırlı verilerin yanı sıra akışlı, sınırsız verileri de temsil edebilir.
- Statik Datasets/DataFrames'e benzer şekilde, akış kaynaklarından akışlı DataFrames/Datasets oluşturmak ve bunlara statik DataFrames/Datasets ile aynı işlemleri uygulamak için ortak giriş noktası SparkSession'ı kullanabilirsiniz.

[Creating streaming DataFrames and streaming Datasets](#)

- Streaming DataFrames, SparkSession.readStream() tarafından döndürülen DataStreamReader arabirimini (Scala/Java/Python belgeleri) aracılığıyla oluşturulabilir. R'de read.stream() yöntemiyle. Statik DataFrame oluşturmaya yönelik okuma arayüzüne benzer şekilde, kaynağın ayrıntılarını (veri formatı, şema, seçenekler vb.) belirtebilirsiniz.

[Input Sources](#)

- File Source (CSV, JSON, ORC, Parquet)
- Kafka Sources
- Socket source (for testing)
- Rate source (for testing)
- Rate Per Micro-Batch source (for testing)

[Schema inference and partition of streaming DataFrames/Datasets](#)

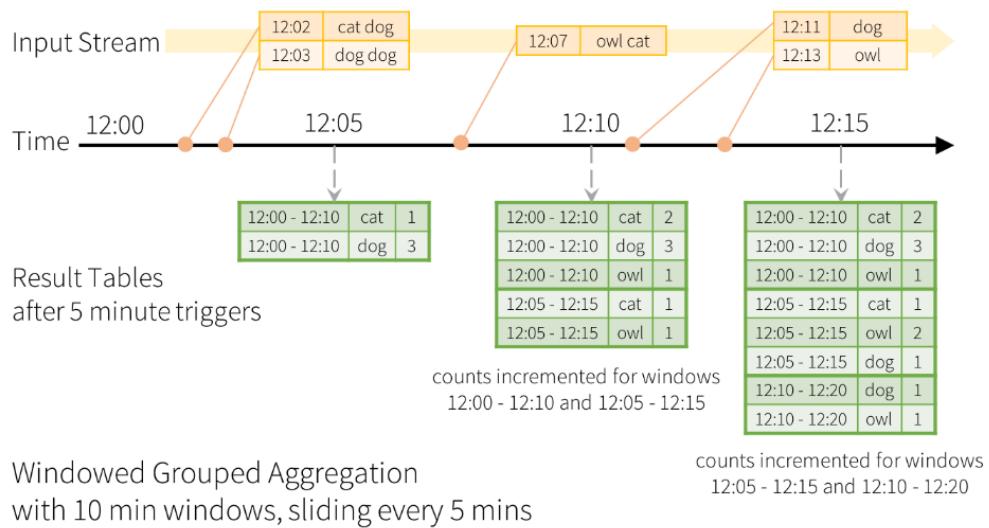
- Varsayılan olarak, dosya tabanlı kaynaklardan Structured Stream, şemayı otomatik olarak çıkarması için Spark'a güvenmek yerine şemayı belirtmenizi gerektirir.

Operations on streaming DataFrames/Datasets

- DataFrame/Dataset streaming'e her türlü işlemi (SQL, RDD işlemleri) uygulayabilirsiniz.

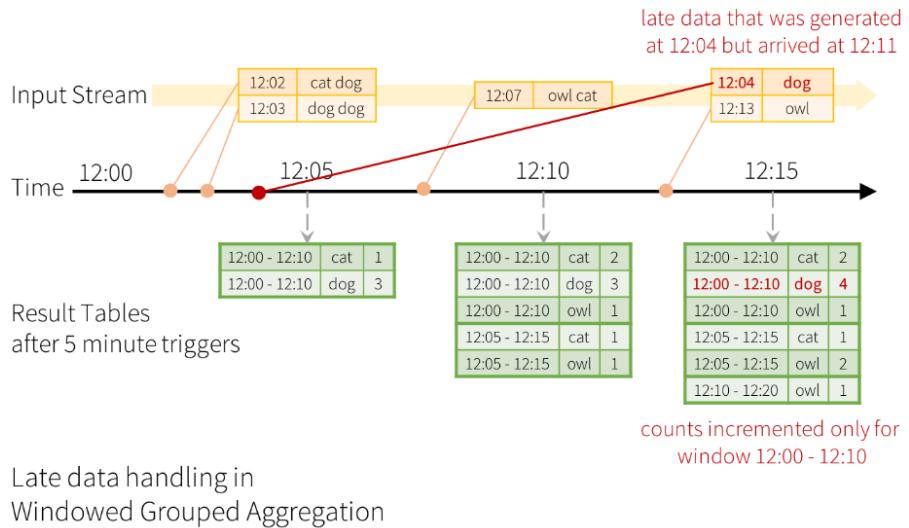
Window Operations on Event Time

- Window-based aggregation, bir satırın olay zamanının düşüğü her pencere için toplam değerler korunur.
- 10 dakikalık pencereler içindeki kelimeleri her 5 dakikada bir güncelleyerek saymak istiyoruz.



Handling Late Data and Watermarking

- Şimdi olaylardan biri uygulamaya geç gelirse ne olacağını düşünün.
- **Watermarking**, motorun verilerdeki mevcut olay zamanını otomatik olarak izlemesine ve buna göre eski durumu temizlemeye çalışmasına olanak tanır.
 - Event timesütununu ve verinin event time açısından ne kadar geç olmasının beklenidine ilişkin eşiği belirterek bir sorgunun watermark tanımlayabilirsiniz.
 - Başka bir deyişle, eşik dahilindeki geç veriler toplanacak, ancak eşikten sonraki veriler düşürmeye başlayacaktır.



Join Operations

- Structured Streaming, akışlı bir Dataset/DataFrame'in statik bir Dataset/DataFrame ile yanı sıra başka bir akışlı Dataset/DataFrame ile birleştirilmesini destekler.
- Akış birleştirmenin sonucu, önceki bölümdeki streaming aggregations sonuçlarına benzer şekilde artımlı olarak oluşturulur.
- İki başlığa ayrılabilir:
 - Stream-Static Joins
 - Stream-Stream Joins
 - Inner Joins with optional Watermarking
 - Outer Joins with Watermarking
 - Semi Joins with Watermarking
 - Support matrix for joins in streaming queries

Streaming Deduplication

- Event'lerdeki benzersiz bir tanımlayıcıyı kullanarak veri akışlarındaki kayıtları tekilleştirebilirsiniz. Bu, benzersiz bir tanımlayıcı sütun kullanılarak statikte tekilleştirmeyle tamamen aynıdır.
- Sorgu, önceki kayıtlardan gerekli miktarda veriyi depolayacak ve böylece yinelenen kayıtları filtreleyebilecektir.
- Aggregation'lara benzer şekilde, watermark'lı veya watermark'sız deduplication kullanabilirsiniz.

Starting Streaming Queries

- DataFrame/Dataset nihai sonucunu tanımladıktan sonra geriye kalan tek şey akış hesaplamasını başlatmaktadır.
- Bunu yapmak için Dataset.writeStream() aracılığıyla döndürülen DataStreamWriter'ı kullanmanız gereklidir.
- Bu arayüzde aşağıdakilerden bir veya birkaçı belirtmeniz gerekecektir.
 - Output sink ayrıntıları: Veri formatı, konum vb.
 - Output Mode: Output sink'e ne yazılacağını belirtin.
 - Query Name: İsteğe bağlı olarak, tanımlama için sorgunun benzersiz bir adını belirtin.
 - Trigger interval
 - Checkpoint location

Output Modes

- Structured Streaming'de çıktı, sorgu için bir mod belirtilerek tanımlanır. Üç çıkış modu mevcuttur:
 - **Complete Mode**
 - Bu modda Spark o ana kadar işlediği tüm satırların çıktısını alacaktır.
 - Her tetiklemeden sonra Sonuç Tablosunun tamamı havuza aktarılacaktır. Bu, toplama sorguları için desteklenir.
 - Bu mod, belirli bir zamanda verilerin tam anlık görüntüsünü oluşturmanız gerekiğinde kullanılmalıdır.
 - **Update Mode**
 - Yalnızca Result Tablosunda son tetikleyiciden bu yana güncellenen satırlar havuza aktarılacaktır.
 - Bu mod, zaman içinde verilerde yapılan değişiklikleri izlemek ve değişikliklerin geçmişini korumak istediğinizde kullanılmalıdır.
 - **Append Mode**
 - Bu, yalnızca son tetikleyiciden bu yana Result Tablosuna eklenen yeni satırların havuza gönderileceği **default** moddur.
 - Bu, yalnızca Sonuç Tablosuna eklenen satırların hiçbir zaman değişimeyeceği sorgular için desteklenir.
 - Dolayısıyla bu mod, her satırın yalnızca bir kez çıktılanacağını garanti eder (fault-tolerance sink varsayılarak).

Output Sinks

- Birkaç tür built-in output sink vardır.
 - File Sink
 - Çıktıyı bir dizine kaydeder.
 - CSV, Json, Parquet, ORC formatlarında yazılabılır.
 - Kafka Sink
 - Çıktıyı Kafka'daki bir veya daha fazla topic'e saklar.
 - Foreach Sink
 - ForeachBatch Sink
 - Console sink (for debugging)
 - Her trigger olduğunda çıktıyı konsola/stdout'a yazdırır.
 - Memory sink (for debugging)
 - Çıktı bellekte in-memory tablo olarak saklanır.

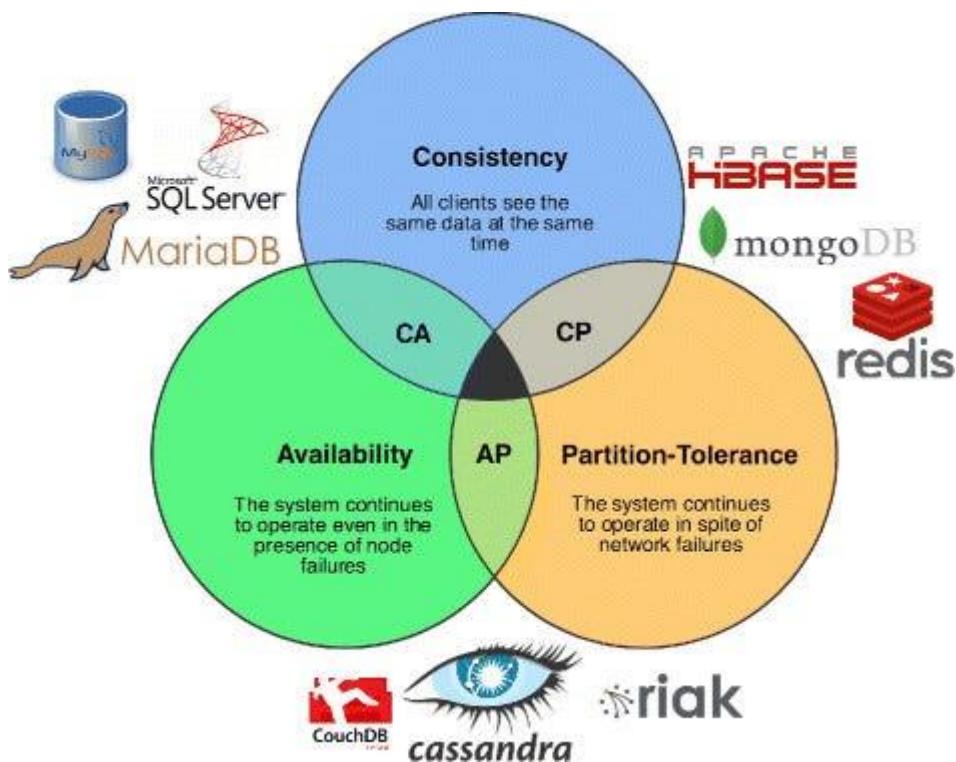
Triggers

- Bir akış sorgusunun tetikleme ayarları, sorgunun sabit bir toplu iş aralığına sahip micro-batch sorgu olarak mı yoksa continuous işleme sorgusu olarak mı yürütüleceğini, akış verisi işlemenin zamanlamasını tanımlar.
- Desteklenen farklı tetikleyici türlerini vardır:
 - unspecified (default)
 - Fixed interval micro-batches
 - Available-now micro-batch
 - Continuous with fixed checkpoint interval

NoSQL

CAP Teoremi

- CAP teoremi, herhangi bir dağıtılmış sistemin veya veri deposunun aynı anda üç garantiden yalnızca ikisini sağlayabileceğini belirtir: tutarlılık (consistency), kullanılabilirlik (availability) ve bölüm toleransı (partition tolerance) (CAP).
 - **Consistency:** Tutarlılık, hangi düğüme bağlanırsa bağlansın tüm istemcilerin aynı verileri aynı anda görmesi anlamına gelir. Bunun gerçekleşmesi için, bir düğüme veri yazıldığında, yazmanın 'başarılı' sayılmasından önce bu verinin sistemdeki diğer tüm düğümlere arasında iletilmesi veya kopyalanması gereklidir.
 - **Availability:** Kullanılabilirlik, bir veya daha fazla düğüm kapalı olsa bile veri talebinde bulunan herhangi bir müşterinin yanıt alması anlamına gelir. Bunu belirtmenin başka bir yolu da, dağıtılmış sisteme tüm çalışan düğümlerin istisnasız olarak herhangi bir istek için geçerli bir yanıt döndürmesidir.
 - **Partition Tolerance:** Bölüm, dağıtılmış bir sisteme ait iletişim kesintisidir; iki düğüm arasındaki bağlantının kaybolması veya geçici olarak gecikmesi. Bölüm toleransı, sisteme ait düğümler arasındaki herhangi bir sayıdaki iletişim kesintisine rağmen kümenin çalışmaya devam etmesi gereği anlamına gelir.



CAP theorem NoSQL database types

- NoSQL veritabanları dağıtılmış ağ uygulamaları için idealdir. Dikey olarak ölçeklenebilir SQL (ilişkisel) muadillerinin aksine, NoSQL veritabanları yatay olarak ölçeklenebilir ve tasarım gereği dağıtilır; birden fazla birbirine bağlı düğümden oluşan büyüyen bir ağ üzerinde hızla ölçeklenebilirler.
 - **CP Database:** Bir CP veritabanı, kullanılabilirlik pahasına tutarlılık ve bölüm toleransı sağlar. Herhangi iki düğüm arasında bir bölüm oluştugunda, sistem, bölüm çözümlenene kadar tutarsız düğümü kapatmalı (yani onu kullanılamaz hale getirmelidir).
 - **AP Database:** Bir AP veritabanı, tutarlılık pahasına kullanılabilirlik ve bölüm toleransı sağlar. Bir bölüm oluştugunda, tüm düğümler kullanılabilir durumda kalır ancak bölümün yanlış ucundakiler, diğerlerinden daha eski bir veri sürümü döndürebilir. (Bölüm çözümlendiğinde AP veritabanları genellikle sistemdeki tüm tutarsızlıklar onarmak için düğümleri yeniden senkronize eder.)
 - **CA Database:** CA veritabanı tüm düğümlerde tutarlılık ve kullanılabilirlik sağlar. Ancak sistemdeki herhangi iki düğüm arasında bir bölüm varsa bunu yapamaz ve bu nedenle hata toleransı sağlayamaz.

NoSQL Veritabanlarına Giriş

- Büyük veri ile birlikte ölçekleme (scaling) ihtiyacından doğmuştur.
- Küçük veri dünyasında veriye çok katı dayatmalarda bulunan SQL'in yetersizliklerine çaredir.
- **Not Only SQL** ya da **No SQL** denir.
- Büyük veri tabanı yönetim sistemlerinin genel adıdır.

RDBMS ACID

- **Atomicity:** Bir işlem bitmeden diğer işlem başlamaz. İşlemin yarıda kesilmesine izin verilmez.
- **Consistency:** Bir işlem veri tabanını tutarsız halde bırakamaz.
- **Isolation:** Bir işlem diğerini etkileyemez, birbirinden yalıtkandır.
- **Durability:** Tamamlanmış bir işlem artık kalıcı olarak saklanır. Uygulama hata alsa veya elektrik kesilse bile işlem sonuçları korunur.

NoSQL BASE

- **Basically Available:** Sistemin CAP Teoremi açısından erişilebilir olduğunu garanti ettiğini gösterir.
- **Soft State:** Sistemin durumunun girdi olmasa bile zaman içinde değişim能力和ını gösterir. Bu Eventual Consistency modelinden kaynaklanıyor.
- **Eventual Consistency:** EC, sistemin girdi alamaması koşuluyla sistemin zaman içinde tutarlı olacağını belirtir.

NoSQL Veritabanlarını Sınıflandırma

Document	Key-Value	XML	Wide-Column	Graph
MongoDB	Redis	BaseX	Big Table	Neo4J
CouchDB	Membase	eXist	Hbase	FlockDB
Elasticsearch	Voldemort		Cassandra	InfineteGraph
CosmosDB	MemcacheDB			
	Kyoto Cabinet			

Feature	Column Oriented	Document Store	Key Value Store	Graph
Table-like schema support (columns)	Yes	No	No	Yes
Complete update/ fetch	Yes	Yes	Yes	Yes
Partial update/ fetch	Yes	Yes	Yes	No
Query/Filter on value	Yes	Yes	No	Yes
Aggregates across rows	Yes	No	No	No
Relationships between entities	No	No	No	Yes
Cross-entity view support	No	Yes	No	No
Batch fetch	Yes	Yes	Yes	Yes
Batch update	Yes	Yes	Yes	No

Document Databases

- Yarı yapısal veri üzerinde CRUD operasyonlarına imkan sağlar.
- Bu kategori altında mevcut olan veri tabanlarının çoğu XML, JSON, BSON veya YAML’yi kullanır. Tipik olarak RESTful API kullanarak http protokolü üzerinden veya çapraz dil birlikte çalışabilirliği için Apache Thrift protokolü üzerinden veri erişimine sahiptir.
- Document, RDBMS’deki bir satırın karşılığıdır.
- Document yarı yapısalıdır. Her iki satır aynı olmak zorunda ancak her iki document değil.
- Primary Key tarzında bir anahtar bilgisi zorunlu değil.
- **Avantajları:**
 - Şema zorunluluğu yok.
 - Zaman içinde gelişebilecek farklı türde içeriklerin depolanmasına ihtiyaç duyulan web tabanlı uygulamalarda çok kullanışlıdır.
 - Birden fazla alanda arama yapmak RDBMS’e göre oldukça kolaydır.
 - Arama bir tabloda değil tüm veri tabanında yapılabilir.

Key-Value Databases

- Document veri tabanlarına benzer.
- Şema dayatması yoktur.
- Değere ulaşmak için anahtar bilme zorunluluğu vardır.
- En yaygın kullanımı, bellek için dağıtılmış veya önbellek içindir.
- **Avantajları:**
 - Anahtara dayalı sorguların optimizasyonu
 - Bellek için veri tabanı kullanım ihtiyacında
 - Caching: Web uygulamaları için daha iyi performans
 - Session Management
 - Leaderboards
 - Inventory
 - Fraud Mitigation
 - Claims Processing

Wide-Column Stores:

EmployeeID	FirstName	LastName	Age	Salary
SM1	Anuj	Sharma	45	10000000
MM2	Anand		34	5000000
T3	Vikas	Gupta	39	7500000
E4	Dinesh	Verma	32	2000000

Row-oriented

SM1,Anuj,Sharma,45,10000000
 MM2,Anand,,34,5000000
 T3,Vikas,Gupta,39,7500000
 E4,Dinesh,Verma,32,2000000

Column-oriented

SM1,MM2,T3,E4
 Anuj,Anand,Vikas,Dinesh
 Sharma,,Gupta,Verma,
 45,34,39,32
 10000000,5000000,7500000,2000000

- **Avantajları:**

- Esnekdir. Öngörülmeyen ihtiyaç ve senaryolara kolay uyum sağlar.
- Yeni bir sütun eklendiğinde varsayılan değer derdi yoktur.
- Sütun bazlı hesaplamalarda performanslıdır.
- Aynı türde veriler ardışık depolandığı için yazma-okuma ve sıkıştırma performansı iyidir.
- Odak noktası uygulamanın sorgulama ihtiyacıdır. Modelleme buna göre yapılır.
- Büyük hacimli veri saklayabilir.

Graph Databases:

- İlişkilerin bir çizge (graph) ile gösterildiği veri tabanı çeşididir.
- Nesneler arasında bağlantı olmalıdır.
- Graph veri tabanları, ilişki ağırlıklı veriler için optimize edilmiş özel amaçlı NoSQL veri tabanları olarak düşünülebilir.

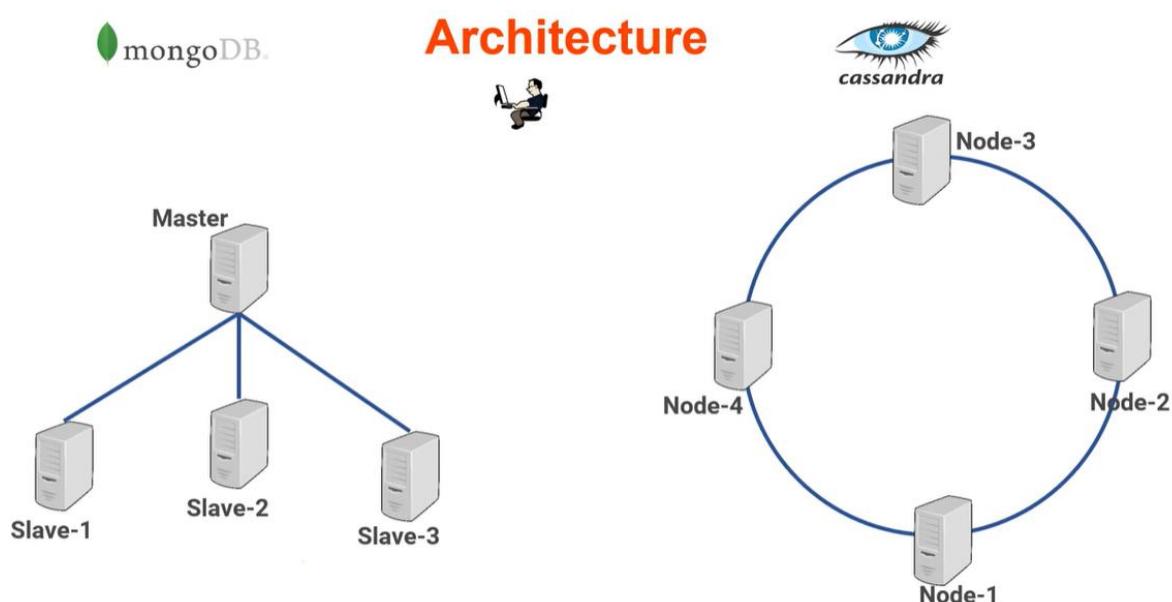
Wide Column: Apache Cassandra

- Cassandra, Amazon Dynamo ve Google Bigtable'dan ilham alan dağıtılmış bir NoSQL veritabanı sistemidir.
- Cassandra, yüksek hacimli, düşük gecikme süreli bulut uygulamaları için özel olarak tasarlanmıştır.
- Öne Çıkan Özellikleri:
 - Açık Kaynaklı
 - Hataya toleranslı
 - Yüksek Performanslı

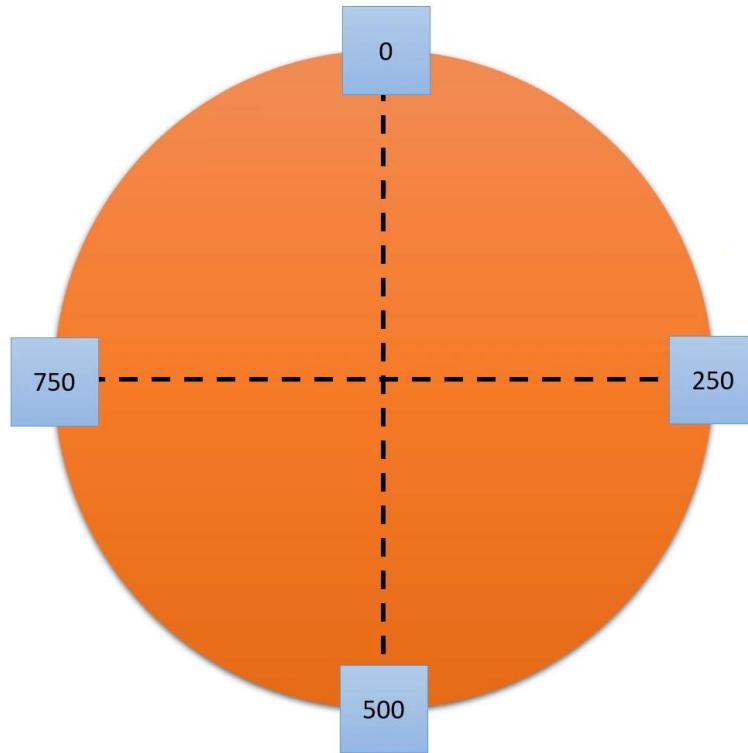
- Dağıtık
- Yüksek ölçeklenebilirliğe sahip
- Tek nokta kırılganlığı yok
- Master-Slave ilişki yok. Bütün node'lar eşit. Bu nedenle bütün node'lar veri isteğine cevap verebilir.
- Zookeeper gibi bir konfigürasyon servisi yok.
- Kullanımı kolay
- Birden fazla veri merkezinde çalışabiliyor.
- RDMS'in bir alternatifi değildir.
- Cassandra bir AP NoSQL veri tabanıdır.
- Cassandra, Java dilinde yazılmıştır. Çalışırken JVM kullanıyor.
- CQL sorgu dilini kullanıyor.

RDBMS ve Cassandra Farklılıkları

- Cassandra'da “join” işlemi yoktur.
- İlişkisel veri tabanı veri odaklı bir modeller yaparken Cassandra sorgu odaklı bir yaklaşım sergiliyor.
 - İlişkisel veri tabanında Data -> Model -> Application iken Cassandra da ise Application -> Model -> Data ise uygulamanın ihtiyaç duyduğu sorgulara göre modelleme yapılır.
- Referential Integrity yok. Dolayısıyla Cascading Delete işlemi de yok.
- Normalizasyon yok aksine Denormalizasyon vardır.



Cassandra Hash Ring



- Master-Slave ilişkisi yoktur.
- Konfigürasyon sunucusuna ihtiyaç yoktur.
- Veri partition key ile halka etrafında dağıtılmıyor.
- Replika oluyor.
- Bütün node'lar isteklere karşılık verebilir.

Cassandra Use Cases

- Sensor Data
- Email App
- E-Commerce
- Fraud Detection in Banks
- Transaction Kayıtlarını tutmak
- Zaman serileri verilerini tutmak
- ...

Document Bases: MongoDB

- Kolay geliştirme yapmak için ve ölçeklenebilir uygulamaları yapmak için tasarlanmıştır.
- Bir kayıt, document olarak adlandırılır ve içerisinde key-value ikilileri vardır.
- MongoDB document'leri JSON objesine çok benzemektedir.

```
{  
    name: "sue",           ← field: value  
    age: 26,              ← field: value  
    status: "A",           ← field: value  
    groups: [ "news", "sports" ] ← field: value  
}
```

- Hızlıdır. Çünkü birbiri ile ilişki olan veriyi bir ara tutuyor.
- Şema dayatması yoktur.
- Sharding (veriyi birden fazla sunucuya dağıtmak) için index gereklidir.
- Geniş ölçekli web uygulamaları için idealdir.
- Öne Çıkan Özellikleri:
 - Yüksek performanslı
 - Gelişmiş bir sorgulama dili var
 - Yüksek erişilebilirlik
 - Yatay Ölçeklenebilir
 - Esnek Şema

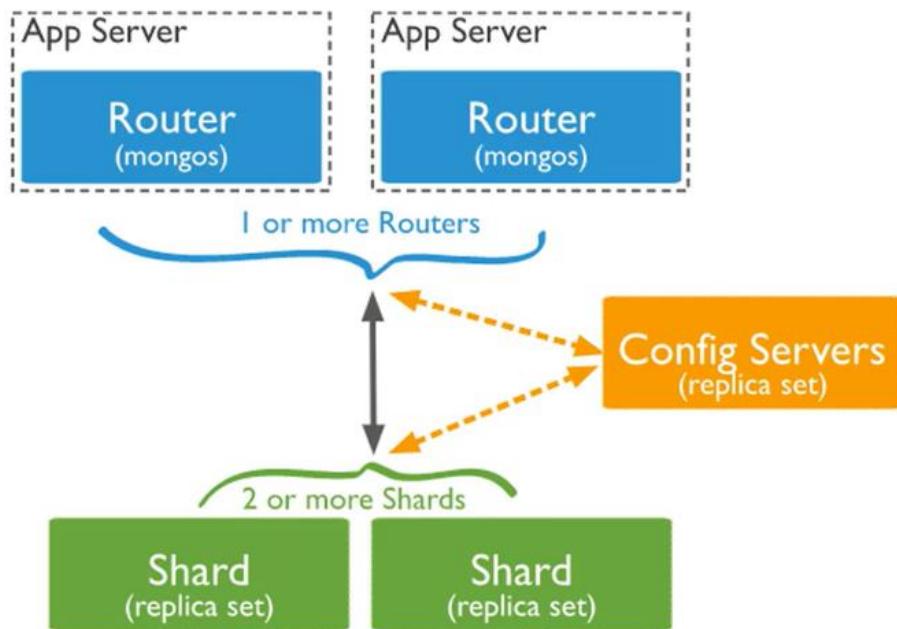
MongoDB vs RDBMS Terminology

RDBMS	MongoDB
Database	Database
Table	Collection
Row	Document
Column	Field
Primary Key	_id
Join	Very Limited

MongoDB Sharding

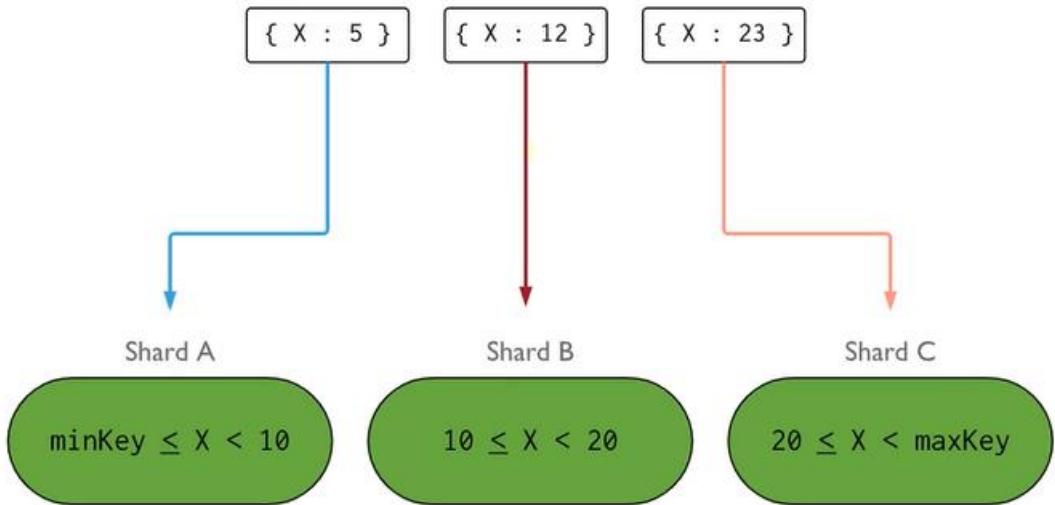
- Veriyi cluster'a dağıtmak için kullanılıyor.
- Yatay ölçeklemeyi mümkün kılıyor.
- Node'lar arasında otomatik load balance sağlıyor.
- Ne kadar mongod instance varsa o kadar yazma kapasitesi vardır.
- Her shard kendi başına bir veri tabanı olarak davranışmaktadır. Birlikte tek bir mantıksal veri tabanı gibi olmaktadır.
- İlave her shard mevcut olanların yükünü azaltmaktadır.
 - 1 TB veri 4 shard için 250 GB, 40 Shard için 25 GB'dır

Shard Cluster



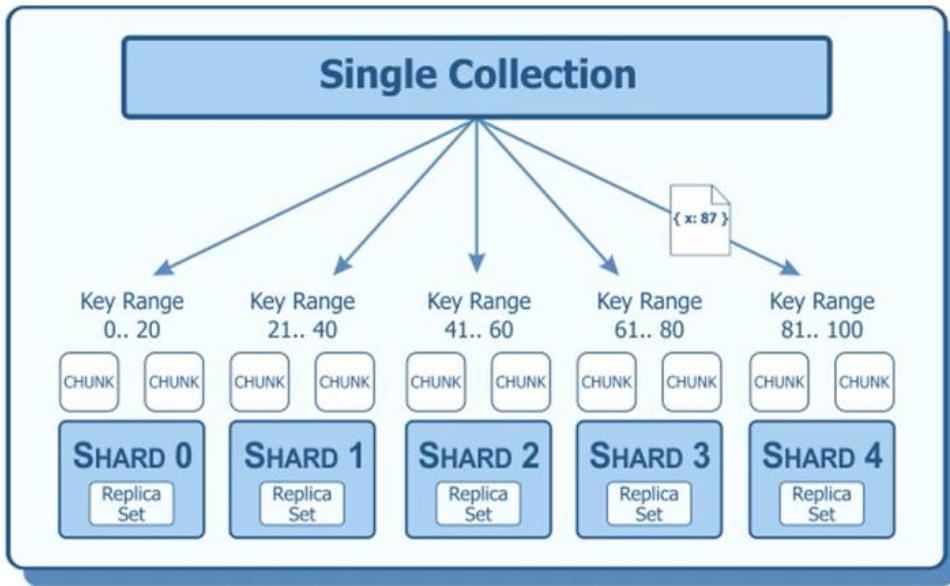
- Shard: Sharded verinin bit alt kümesidir. Her bir shard replica set'leri ile dağıtılabılır.
- Mangos: Sorguların Shard'lara yönlendirmesini sağlar.
- Config Server: Cluster'şar için Metadata'ları ve konfigürasyon ayalarını saklar.

Shard Keys:



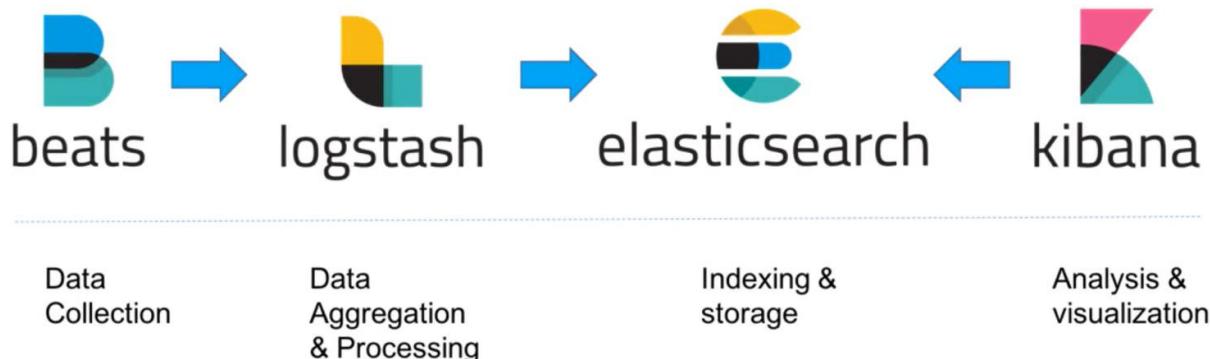
- MongoDB, Collection'ların document'lerini Shard Keys aracılığıyla shard'lara dağıtır.
- Shard key, vir field'dan veya hedef collection document'lerinin hepsinde bulunan bir field'den oluşur.

Chunks



- MongoDB, parçalanmış verileri chunk'lara ayırır. Her chunk'in, shard key'e dayalı olarak kapsayıcı bir alt ve özel bir üst aralığı vardır.
- MongoDB'de varsayılan chunk boyutu 64 MB'dır.

Full-Text Search: Elasticsearch



- Apache Lucene'nin devamı niteliğindedir.
- Yatay ölçeklenebilir.
- Full-text arama sunmaktadır.
 - Bunun yanında, aggregation sorguları gibi sorgular da yapılabilmektedir.
- Doğru kullanıldığında Spark ve Flink gibi birçok Big Data aracından daha hızlı çalışabilir.
- RESTApi ile çalışabilmektedir.
- Bir arayüzü yoktur.
 - http komutlarının gönderildiği her araç sayesinde iletişim kurulabilir.
- Daha gelişmiş bir arayüz ve görselleştirme için Kibana ile kullanılabilir.
- JSON dosya yapısı ile çalışmaktadır.
- Varsayılan portu 9200'dür.

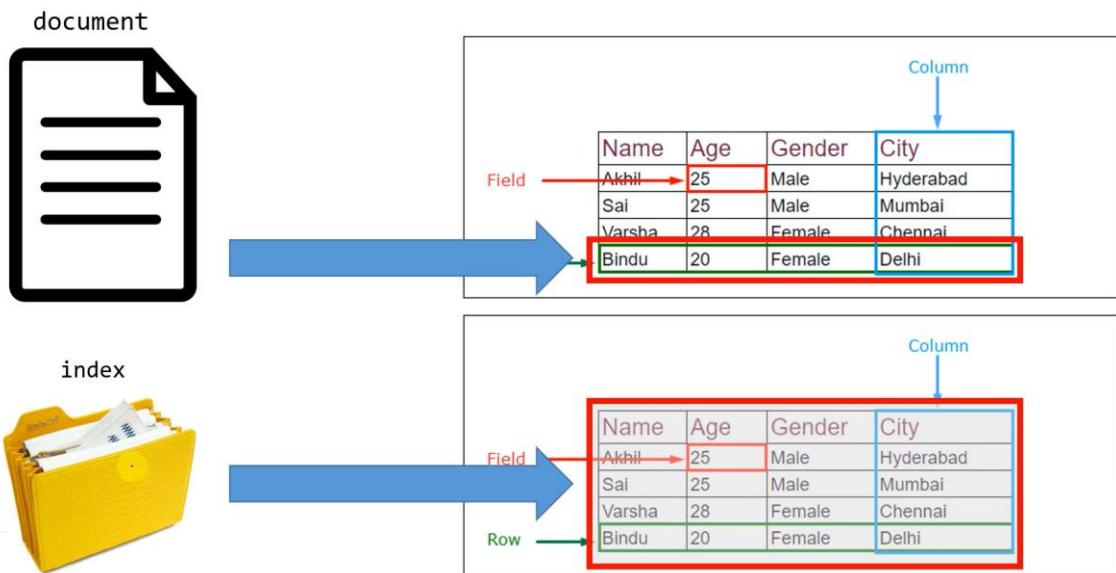
Elasticsearch Cornerstone:

Document

- JSON formatında bilgileri tutan bir veri yapısıdır.
- Klasik bir veri tabanındaki satırı karşılık gelmektedir.
- Her dokümanın eşsiz bir ID'si vardır.
- Bir key verilmez ise Elasticsearch bunu kendisi üretmektedir.
- Dokümanlar güncellenebilir.
 - Dokümanlar immutable'dır.
 - Belgeler güncellendiğinde versiyonlanır. Eski versiyonlar bir süre saklanır daha sonra silinir.
- Bir dokümani veya bütün index'i silmek mümkündür.

Index

- ES'de sorgulanabilen en büyük varlıktır.
- İlişkisel/Klasik veri tabanındaki tabloya benzetilebilir
- Her index'in bir şeması vardır.
 - Şema ise dokümanın hangi türlere sahip olduğunu tanımlamaktadır.



Inverted Index

Document-1

- ✓ Hello we learn big data in this course.

Document-2

- ✓ Elasticsearch doesn't dependent on Hadoop and it is popular big data platform.

Inverted index

big:	1,2
data:	1
hadoop:	1,2
elasticsearch:	2
.....	

Data Pipelines and Workflow Scheduling

Apache Airflow

- Apache Airflow, batch-oriented workflows geliştirmeye, planlamaya ve izlemeye yönelik açık kaynaklı bir platformdur.

Neden Airflow?

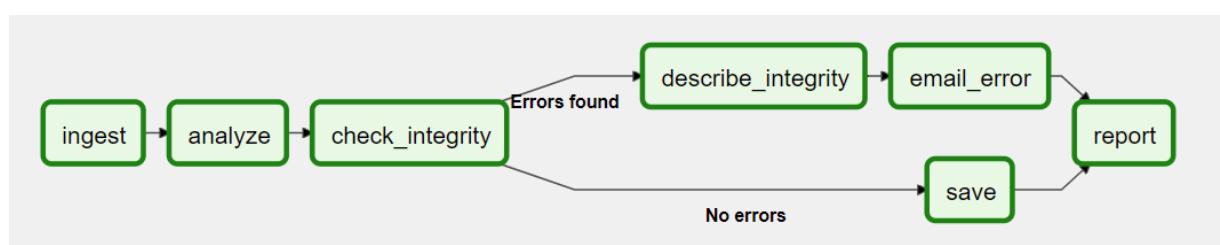
- Apache Airflow, iş akışlarını programlı olarak yazmaya, planlamaya ve izlemeye yönelik bir platformdur. Özellikle karmaşık data pipelines oluşturmak ve düzenlemek için kullanışlıdır.
- Veri düzenleme, herhangi bir modern veri yiğininin kalbinde yer alır ve data pipeline'ların ayrıntılı otomasyonunu sağlar. Orkestrasyonla, data pipeline'daki eylemler birbirinden haberdar olur ve veri ekibiniz iş akışlarını izlemek, düzenlemek ve sorunlarını gidermek için merkezi bir konuma sahip olur.

Airflow'un Faydalari

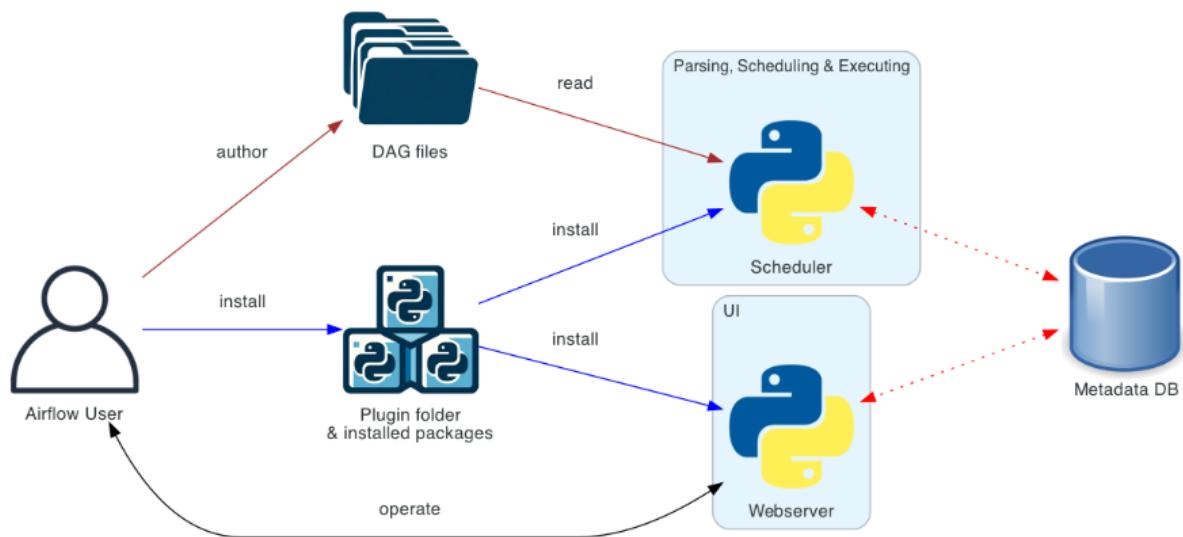
- Dynamic data pipelines
- CI/CD for data pipelines
- Flexibility, Extensibility, and Scalability
- Cloud-Native
- Fault Tolerant
- Job Schedule, Monitor and Alert

Airflow Architecture

- Airflow, iş akışları oluşturmanıza ve çalıştırmanıza olanak tanıyan bir platformdur.
- Bir iş akışı, DAG (Directed Acyclic Graph) olarak temsil edilir ve bağımlılıklar ve veri akışları dikkate alınarak düzenlenmiş, Tasks adı verilen ayrı iş parçalarını içerir.



- DAG, görevlerin yürütülme sırasını tanımlayan görevler arasındaki bağımlılıkları belirtir. Tasks, veri almak, analiz yürütmek, diğer sistemleri tetiklemek veya daha fazlası gibi ne yapılacağını açıklar.



Airflow Components

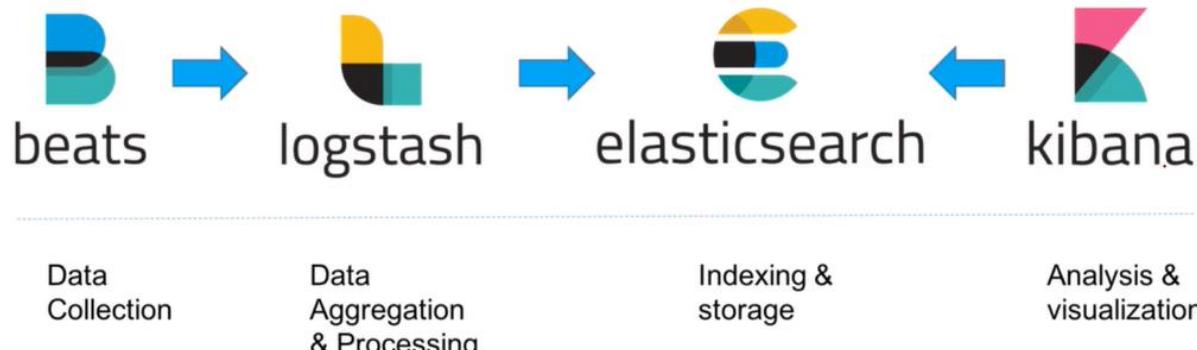
- Gerekli Bileşenler:
 - Scheduler
 - Webserver
 - Database
 - Executor
- Opsiyonel Bileşenler:
 - Worker
 - Triggerer
 - Dag Processor

Executor Types

- Local Executors
 - Local Executor
 - Sequential Executor
- Remote Executors
 - Celery Executor
 - CeleryKubernetes Executor
 - Dask Executor
 - Kubernetes Executor
 - LocalKubernetes Executor

Logstash

- Açık kaynaklı veri işleme motorudur.
- Bir sürü kaynak ve hedefleri desteklemektedir.
- Log dosyalarına ek olarak XML, HTML ve csv dosyalarına işleyebiliyor.
- Yatay olarak ölçeklenebiliyor.



Logstash Pipeline

- Verileri işlemek için plugin'leri (inputs, outputs, filters ve bazen codec'ler) bir araya getirerek bir pipeline oluşturabilirsiniz.
 - Input Plugins
 - azure_event_hubs, beats, cloudwatch, elasticsearch, github, http ...
 - Output Plugins
 - Csv, app_search, boundary, cloudwatch, http, kafka ...
 - Filters
 - Age, alter, aggregate, csv, bytes ...
- Çok basit bir pipeline yalnızca bir input ve bir output içerebilir. Çoğu işlem hattı en az bir filtre eklentisi içerir çünkü ETL (çıkarma, dönüştürme, yükleme) "dönüştürme" kısmı burada gerçekleşir.

Apache NiFi

- NiFi, sistemler arasındaki veri akışını otomatikleştirmek için tasarlandı.
- Ölçeklenebilir
- Dağıtık
- Hata Toleransı
- Düşük gecikme süresi
- Yüksek verim
- Dinamik önceliklendirme
- Back pressure
- Güvenli

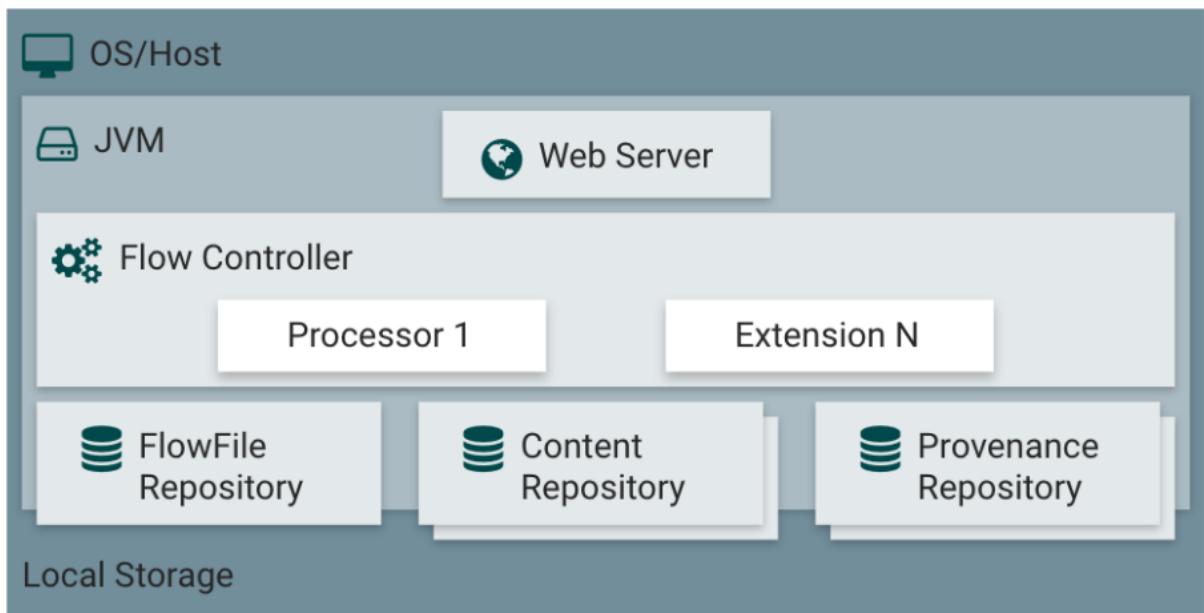
Apache NiFi Temel Bileşenleri

- NiFi'nin temel tasarım konseptleri, Flow Based Programming (fbp) ana fikirleriyle yakından ilgilidir.
- NiFi'yi bilgi akışını bir dizi olay olarak ele alan bir şey olarak düşünmeye çalışın.
 - Dosya geliyor/alınıyor, dosya okunuyor, dosya değiştiriliyor, dosya başka bir yere taşınıyor.
 - Basit bir ifadeyle, bir dosyadan, dosyayı okumaya ve değiştirmeye yarayan bir araçtan ve dosyayı başka bir yere yönlendirmeye yarayan bir araçtan oluşur.

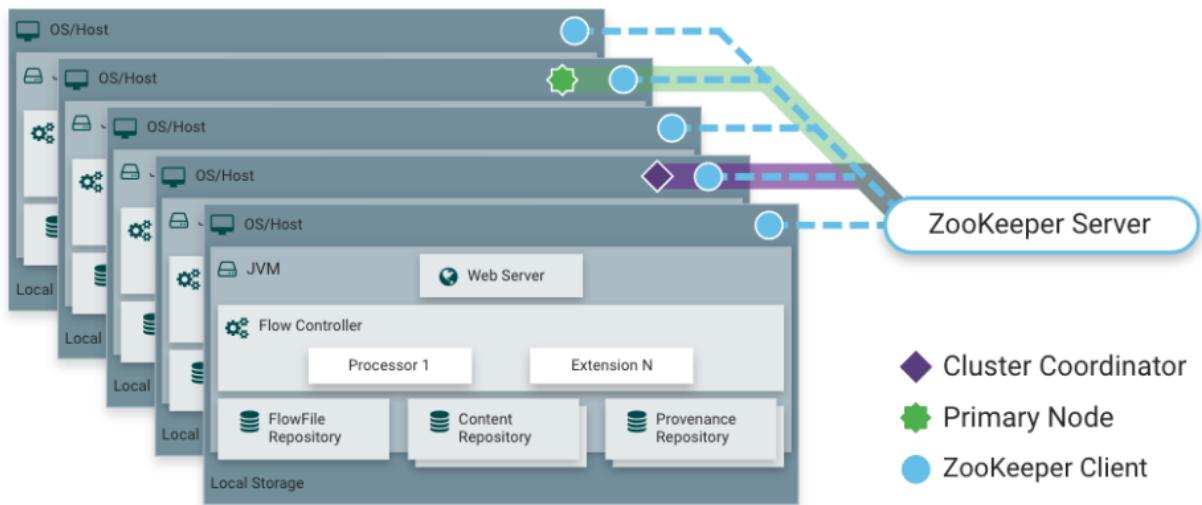
NiFi Term	FBP Term	Description
FlowFile	Information Packet	A FlowFile represents each object moving through the system and for each one, NiFi keeps track of a map of key/value pair attribute strings and its associated content of zero or more bytes.
FlowFile Processor	Black Box	Processors actually perform the work. In [eip] terms a processor is doing some combination of data routing, transformation, or mediation between systems. Processors have access to attributes of a given FlowFile and its content stream. Processors can operate on zero or more FlowFiles in a given unit of work and either commit that work or rollback.
Connection	Bounded Buffer	Connections provide the actual linkage between processors. These act as queues and allow various processes to interact at differing rates. These queues can be prioritized dynamically and can have upper bounds on load, which enable back pressure.
Flow Controller	Scheduler	The Flow Controller maintains the knowledge of how processes connect and manages the threads and allocations thereof which all processes use. The Flow Controller acts as the broker facilitating the exchange of FlowFiles between processors.
Process Group	subnet	A Process Group is a specific set of processes and their connections, which can receive data via input ports and send data out via output ports. In this manner, process groups allow creation of entirely new components simply by composition of other components.

Apache NiFi Architecture

- Apache NiFi'nin JVM makinesinde çalışan bir processor, flow controller ve web server'ı vardır. Ek olarak, şekilde gösterildiği gibi FlowFile repository, Content repository ve Provenance repository olmak üzere üç repository içerir.

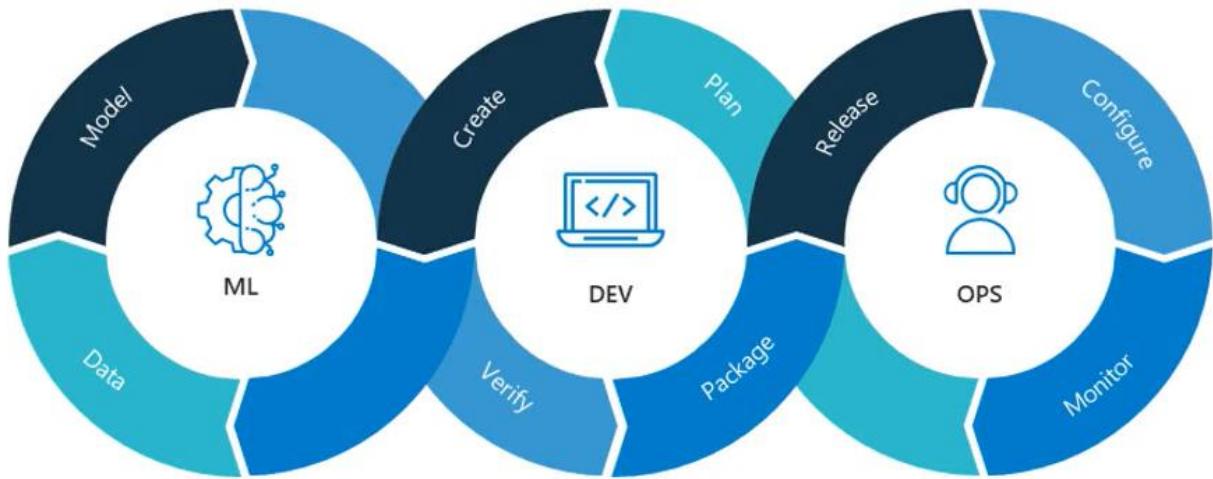


- NiFi aynı zamanda bir küme içerisinde de çalışabilmektedir.



Container Environments and Deploying Machine Learning Models

MLOPS



- MLOps, Makine Öğrenimi İşlemleri anlamına gelir.
- MLOps, Makine Öğrenimi mühendisliğinin temel bir işlevidir ve makine öğrenimi modellerini üretime alma, ardından bunların bakımı ve izlenmesi sürecini kolaylaştırmaya odaklanır.

MLOPS Maturity Model

- Olgunluk modeli, üretim düzeyinde makine öğrenimi uygulama ortamının oluşturulması ve işletilmesindeki sürekli gelişmeyi gösterir.
 - İşlerin ne kadar optimize edildiğine bakar. Ne kadar optimize edilmiş ise Level'1 artar.
- Beş seviye bulunmaktadır:
 - Level 0
 - Level 1
 - Level 2
 - Level 3
 - Level 4

Level	Description	Highlights	Technology
0	No MLOps	<ul style="list-style-type: none"> Difficult to manage full machine learning model lifecycle The teams are disparate and releases are painful Most systems exist as "black boxes," little feedback during/post deployment 	<ul style="list-style-type: none"> Manual builds and deployments Manual testing of model and application No centralized tracking of model performance Training of model is manual
1	DevOps but no MLOps	<ul style="list-style-type: none"> Releases are less painful than No MLOps, but rely on Data Team for every new model Still limited feedback on how well a model performs in production Difficult to trace/reproduce results 	<ul style="list-style-type: none"> Automated builds Automated tests for application code
2	Automated Training	<ul style="list-style-type: none"> Training environment is fully managed and traceable Easy to reproduce model Releases are manual, but low friction 	<ul style="list-style-type: none"> Automated model training Centralized tracking of model training performance Model management
3	Automated Model Deployment	<ul style="list-style-type: none"> Releases are low friction and automatic Full traceability from deployment back to original data Entire environment managed: train > test > production 	<ul style="list-style-type: none"> Integrated A/B testing of model performance for deployment Automated tests for all code Centralized tracking of model training performance
4	Full MLOps Automated Operations	<ul style="list-style-type: none"> Full system automated and easily monitored Production systems are providing information on how to improve and, in some cases, automatically improve with new models Approaching a zero-downtime system 	<ul style="list-style-type: none"> Automated model training and testing Verbose, centralized metrics from deployed model

MLFlow

- Makine Öğrenimi (ML) dünyasına adım atmak heyecan verici bir yolculuktur ancak çoğu zaman inovasyonu ve denemeyi engelleyebilecek karmaşıklıkları da beraberinde getirir.
- MLflow, bu dinamik ortamda bu sorunların çoğuna yönelik bir çözüm olup, makine öğrenimi yaşam döngüsünü kolaylaştırmak ve makine öğrenimi uygulayıcıları arasındaki işbirliğini teşvik etmek için araçlar sunar ve süreçleri basitleştirir.

Core Components of MLflow

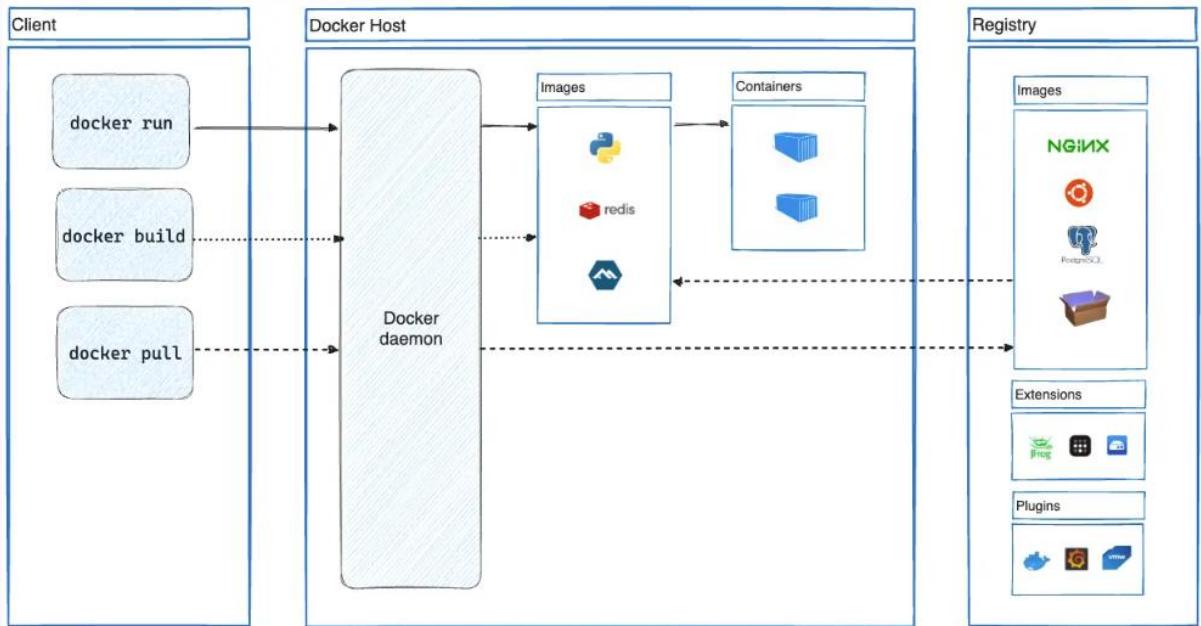
- MLflow, özünde, ML iş akışını basitleştirmeyi amaçlayan bir araç paketi sağlar. Makine öğrenimi geliştirme ve dağıtımının çeşitli aşamalarında makine öğrenimi uygulayıcılarına yardımcı olmak üzere tasarlanmıştır. Kapsamlı tekliflerine rağmen MLflow'un işlevleri çeşitli temel bileşenlere dayanmaktadır:
 - Tracking
 - Model Registry
 - MLflow Deployments for LLMs
 - Evaluate
 - Prompt Engineering UI
 - Recipes
 - Projects

Docker

- Docker, uygulamaları geliştirmek, göndermek ve çalıştırmak için açık bir platformdur. Docker, yazılımlarınızı hızlı bir şekilde teslim edebilmeniz için uygulamalarınızı altyapınızdan ayıranızıza olanak tanır.
- Docker, bir uygulamayı container adı verilen gevşek bir şekilde yalıtılmış bir ortamda paketleme ve çalıştırma olanağı sağlar.
- Containers hafiftir ve uygulamayı çalıştırmak için gereken her şeyi içerir; dolayısıyla ana bilgisayarda yüklü olanlara güvenmeniz gerekmek.

Docker Mimarisi

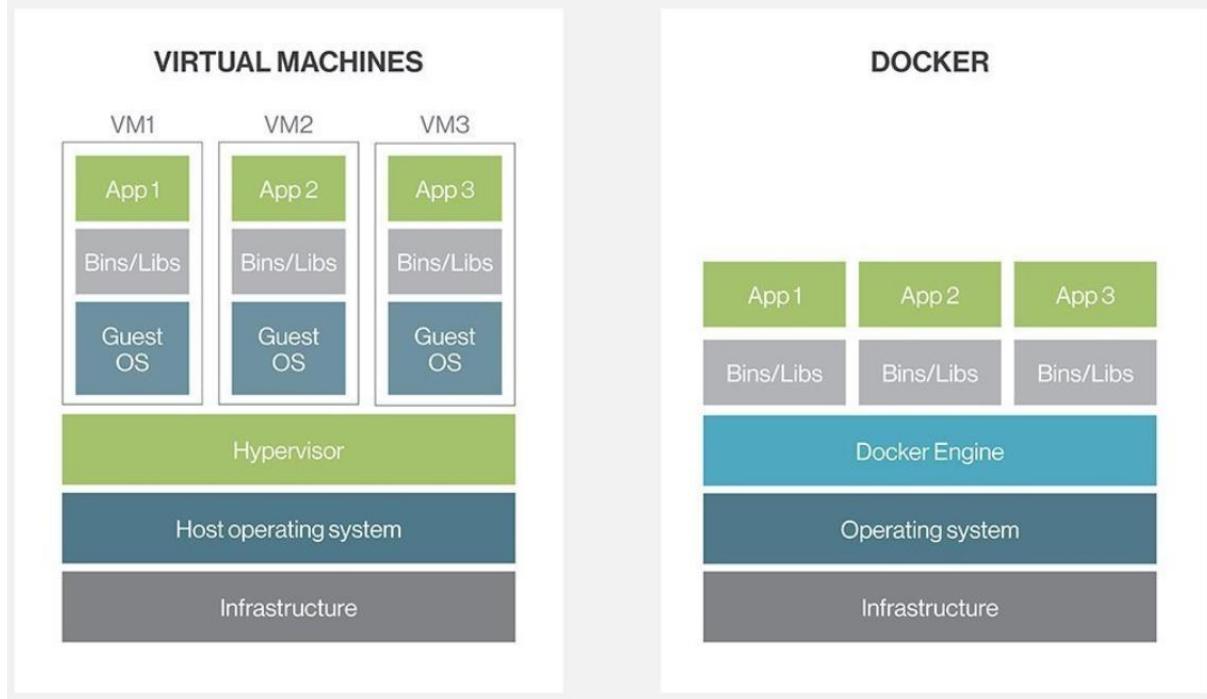
- Docker istemci-sunucu mimarisini kullanır. Docker daemon, Docker konteynerlerinizi oluşturma, çalışma ve dağıtma işlerinin ağır yükünü üstlenen Docker arka plan programıyla konuşur.
- Docker daemon ve arka plan programı aynı sistemde çalışabilir veya bir Docker daemon'ı uzak bir Docker arka plan programına bağlayabilirsiniz.



Docker'in Avantajları ve Dezavantajları

- Docker, Docker ana bilgisayarları genelinde konteynerleri hızlı bir şekilde oluşturmak, oluşturmak, dağıtmak, ölçeklendirmek ve denetlemek için fiili standart bir platform olarak ortaya çıktı.
- Etkin konteynerleştirilmiş uygulama geliştirmenin yanı sıra Docker'in diğer avantajları arasında aşağıdakiler yer alır:
 - Kullanıcıların çeşitli ana bilgisayarlar üzerinden kapsayıcıları kaydedip paylaşılmasını sağlayan yüksek derecede taşınabilirlik.
 - Daha düşük kaynak kullanımı
 - Virtual Machine'lere kıyasla daha hızlı dağıtım.
- Docker'in potansiyel zorlukları da var:
 - Bir enterprise'da mümkün olan konteyner sayısının verimli bir şekilde yönetilmesi zor olabilir.
 - Konteyner kullanımı, ayrıntılı sanal barındırmadan uygulama bileşenlerinin ve kaynaklarının orkestrasyonuna doğru evriliyor. Sonuç olarak, yüzlerce geçici konteyneri içerebilen bileşenleştirilmiş uygulamaların dağıtımları ve ara bağlantısı büyük bir engel haline geliyor.

Virtual machines versus Docker



Kubernetes

- k8s" veya "kube" olarak da bilinen Kubernetes, konteynerli uygulamaların dağıtımını, yönetimini ve ölçeklendirilmesini planlamak ve otomatikleştirmek için kullanılan bir konteyner düzenleme platformudur.
- Kubernetes, 2014 yılında açık kaynak haline getirilmeden önce ilk olarak Google'daki mühendisler tarafından geliştirildi.

Kubernetes Ne Yapar?

- Kubernetes, aşağıdakiler de dahil olmak üzere uygulama yaşam döngüsü boyunca konteynerle ilgili görevleri planlar ve otomatikleştirir:
 - Deployment
 - Rollouts
 - Service discovery
 - Storage provisioning
 - Load balancing
 - Autoscaling
 - Self-healing for high availability

Kubernetes Cluster Mimarisi

