CLOUDERA
Educational Services

Cloudera Data Engineering: Developing Applications with Apache Spark

CLOUDERA

Introduction

Chapter 1

Course Chapters

- **Introduction**
- Why Data Engineering
- Introduction to Zeppelin
- HDFS Introduction
- YARN Introduction
- Distributed Processing History
- Working With RDDs
- Working With Data Frames
- Introduction to Apache Hive
- Transforming Data with Hive
- Data Engineering with Hive
- Hive Spark Integration
- Distributed Processing Challenges
- Spark Distributed Processing
- Spark Distributed Persistence
- Working with the Data Engineering Service
- Working with Airflow
- Workload Manager Introduction
- Conclusion
- Appendix: Working with Datasets in Scala

Trademark Information

- **The names and logos of Apache products mentioned in Cloudera training courses, including those listed below, are trademarks of the Apache Software Foundation**

Apache Accumulo	Apache Hadoop	Apache NiFi	Apache Spark
Apache Avro	Apache HBase	Apache Oozie	Apache Sqoop
Apache Airflow	Apache Hive	Apache ORC	Apache Tez
Apache Atlas	Apache Impala	Apache Parquet	Apache Zeppelin
Apache Bigtop	Apache Kafka	Apache Phoenix	Apache ZooKeeper
Apache Druid	Apache Knox	Apache Ranger	
Apache Flink	Apache Kudu	Apache Solr	

- **All other product names, logos, and brands cited herein are the property of their respective owners**

Chapter Topics

Introduction

- **About This Course**
- Introductions
- About Cloudera
- About Cloudera Educational Services
- Course Logistics

About This Course

- **During this course you will learn**
  - How the Apache Hadoop ecosystem fits in with the data processing lifecycle
  - How data is distributed, stored, and processed in a Hadoop cluster
  - How to write, configure, and deploy Apache Spark applications on a Data Engineering service
  - How to use the Spark and/or Hive to explore, process, and analyze distributed data
  - How to query data using Spark SQL, DataFrames, and Datasets
  - How to chain your Spark and Hive applications using AirFlow
  - How to monitor and troubleshoot your applications using the Workload Manager

Agenda

Day 1	Day 2	Day 3	Day 4
Class Introduction	Spark DataFrames	Apache Hive Introduction	Distributed Processing
Zeppelin Introduction	Reading DataFrames	Transforming data with Hive	Distributed Persistence
HDFS Introduction	Working with Columns	Data Engineering with Hive	Working the Data Engineering Service
YARN Introduction	Transforming DataFrames	• Partitions • Buckets • Skewed Tables	Working with Airflow
Distributed Processing History	Working with UDFs	• Text Data • Complex Types	Working with WXM
Spark RDDs	Working with Windows	Spark Integration with Hive	(*) Appendix: Scala Datasets

(*) if time allows

Chapter Topics

Introduction

- About This Course
- **Introductions**
- About Cloudera
- About Cloudera Educational Services
- Course Logistics

Introductions

- **About your instructor**

- **About you**
  - Currently, what do you do at your workplace?
  - What is your experience with database technologies, programming, and query languages?
  - What is your experience with Spark and Big Data?
  - What do you expect to gain from this course? What would you like to be able to do at the end that you cannot do now?

Chapter Topics

Introduction

- About This Course
- Introductions
- **About Cloudera**
- About Cloudera Educational Services
- Course Logistics

About Cloudera

CLOUDERA

THE HYBRID DATA COMPANY

We believe that data can make what is impossible today, possible tomorrow

We empower people to transform complex data anywhere into actionable insights faster and easier

We deliver a hybrid data platform with secure data management and portable cloud-native data analytics

Future Data Ecosystem

"The future data ecosystem should leverage distributed data management components — which may run on multiple clouds and/or on-premises — but are treated as a cohesive whole with a high degree of automation. Integration, metadata and governance capabilities glue the individual components together."

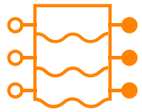
Gartner®

Strategic Roadmap for Migrating Data Management to the Cloud
Published 21 March 2022 - ID G00746011, Analyst(s): Robert Thanaraj, Adam Ronthal, Donald Feinberg

A More Specific Definition of Hybrid

Hybrid = Freedom to move existing and future applications, data, and users bi-directionally between the data center and multiple public clouds

Applications include streams, pipelines, workloads, workspaces, and other data products

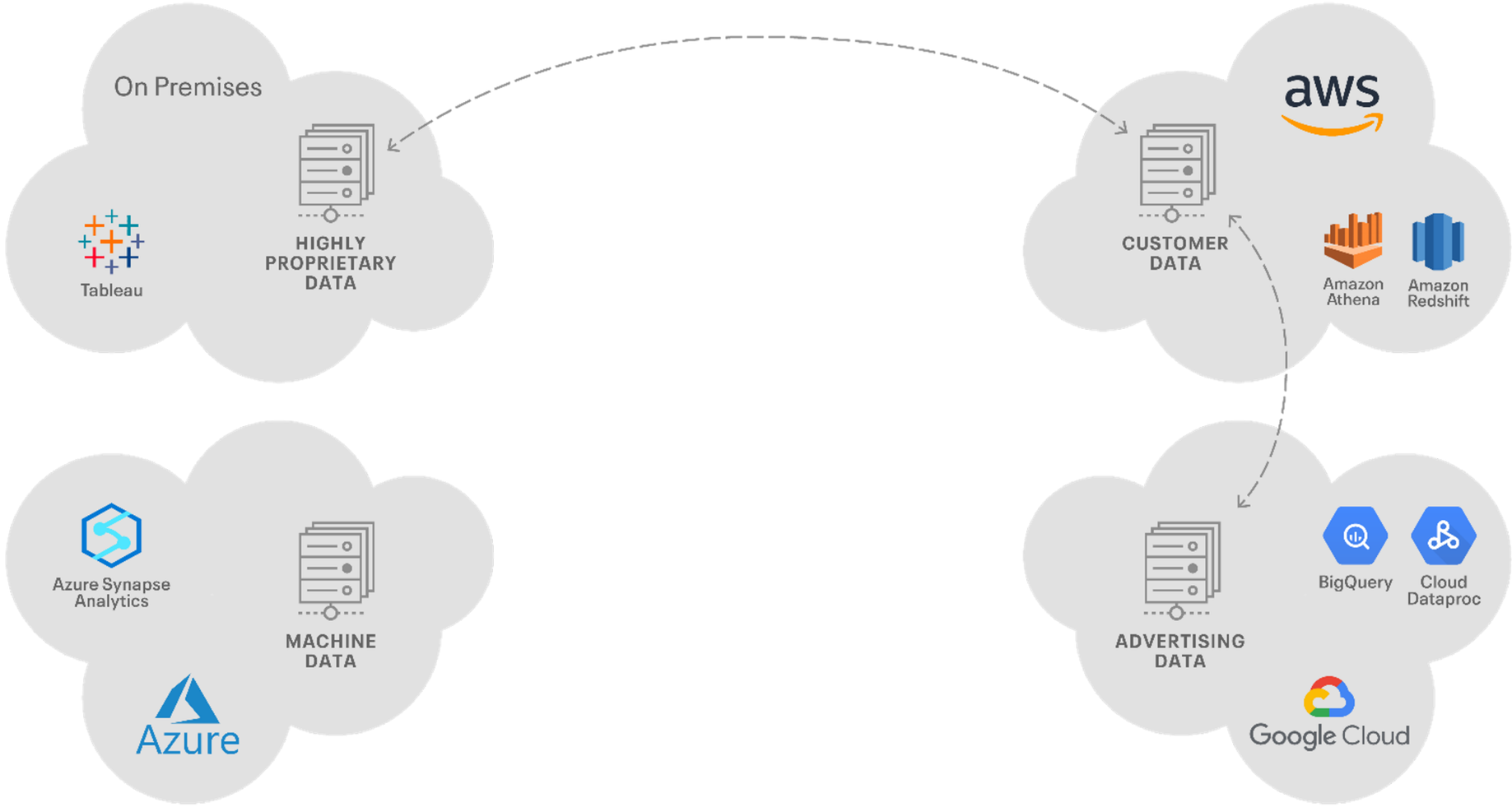


Data considerations include formats, policies, metadata, replication etc.

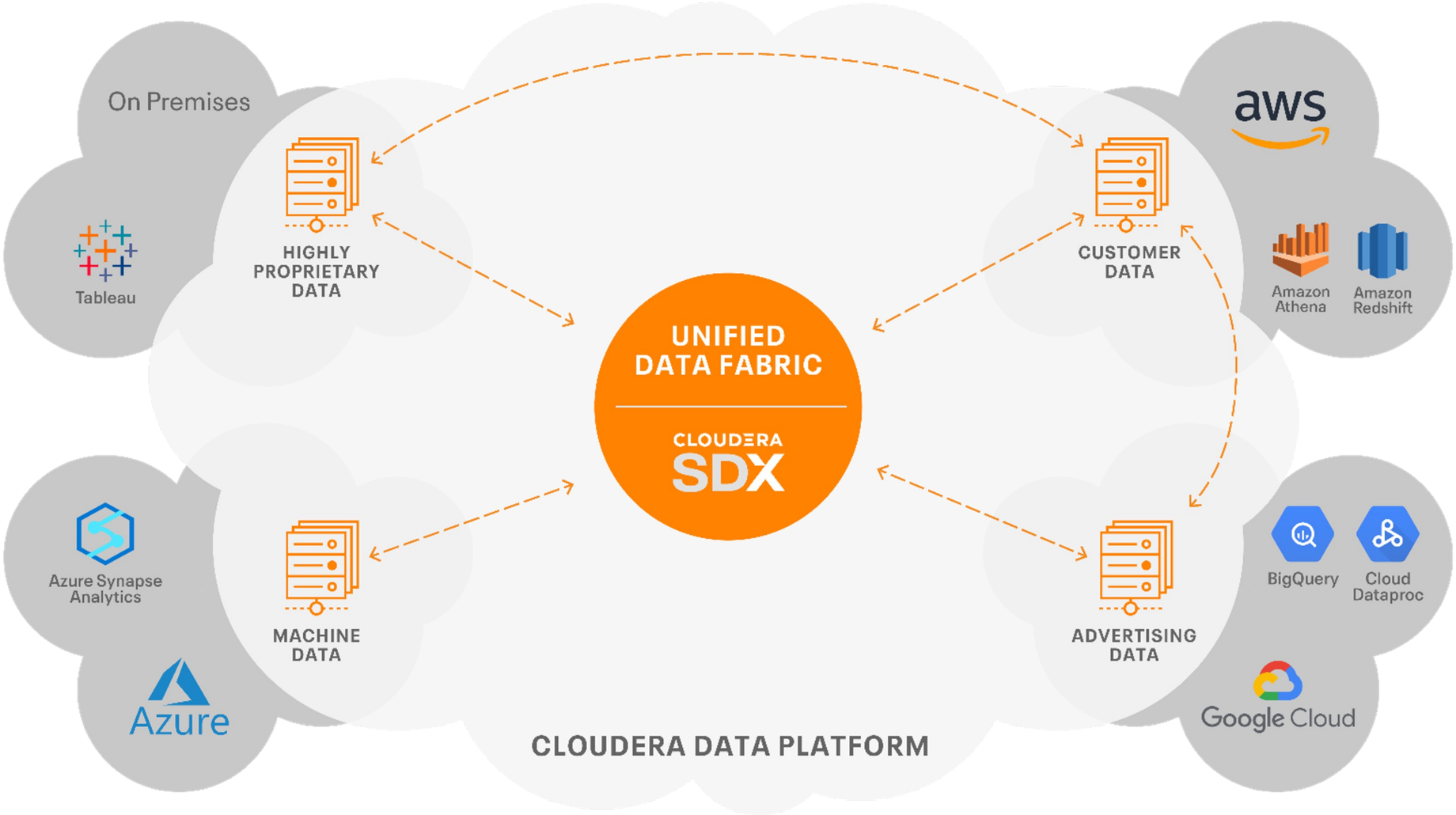
Users need to leverage existing skills and processes, while continuously modernizing

Cloudera's Hybrid Data Platform

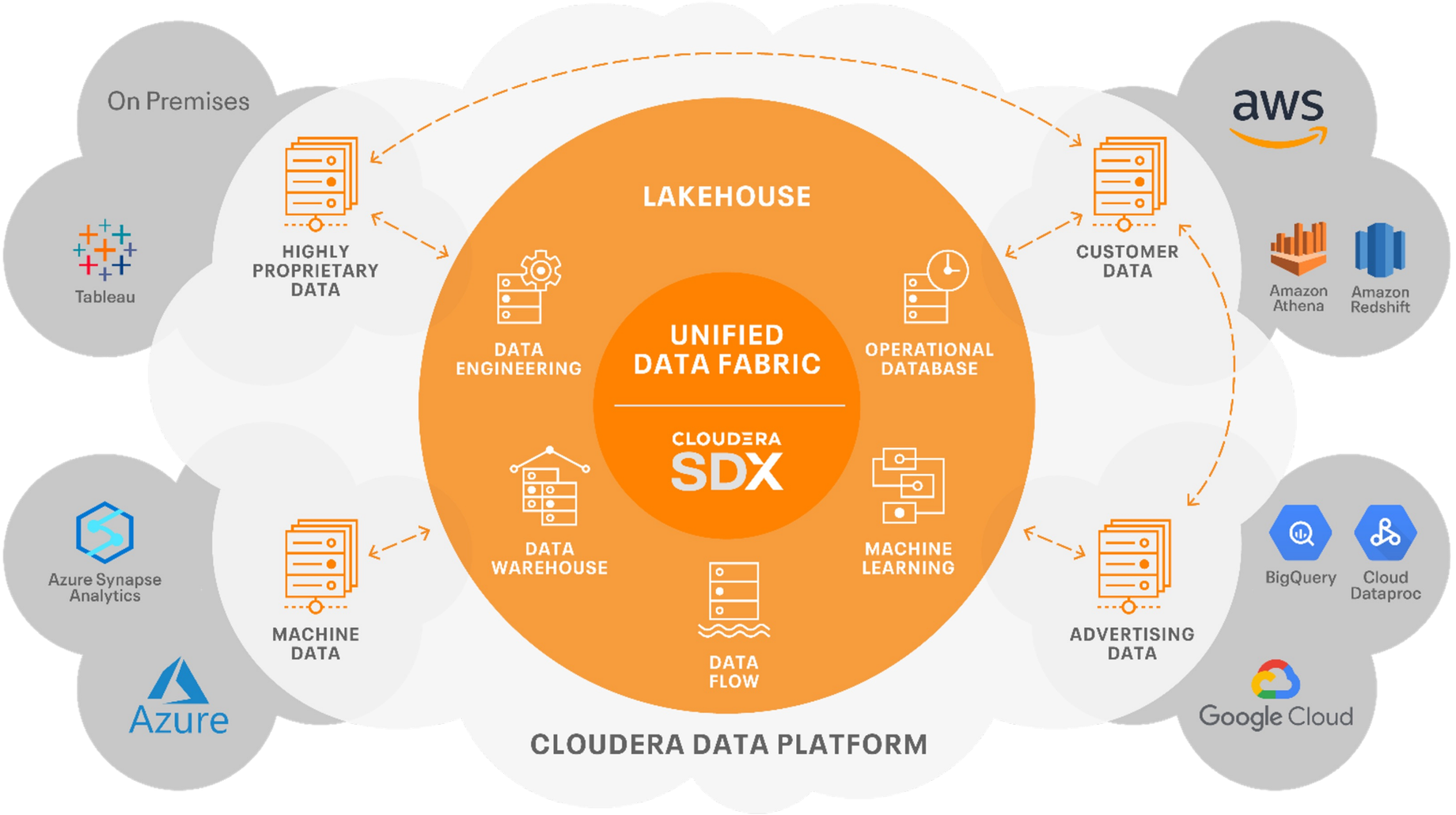
Portable – Interoperable – Open – Secure



Cloudera's Hybrid Data Platform (2)



Cloudera's Hybrid Data Platform (3)

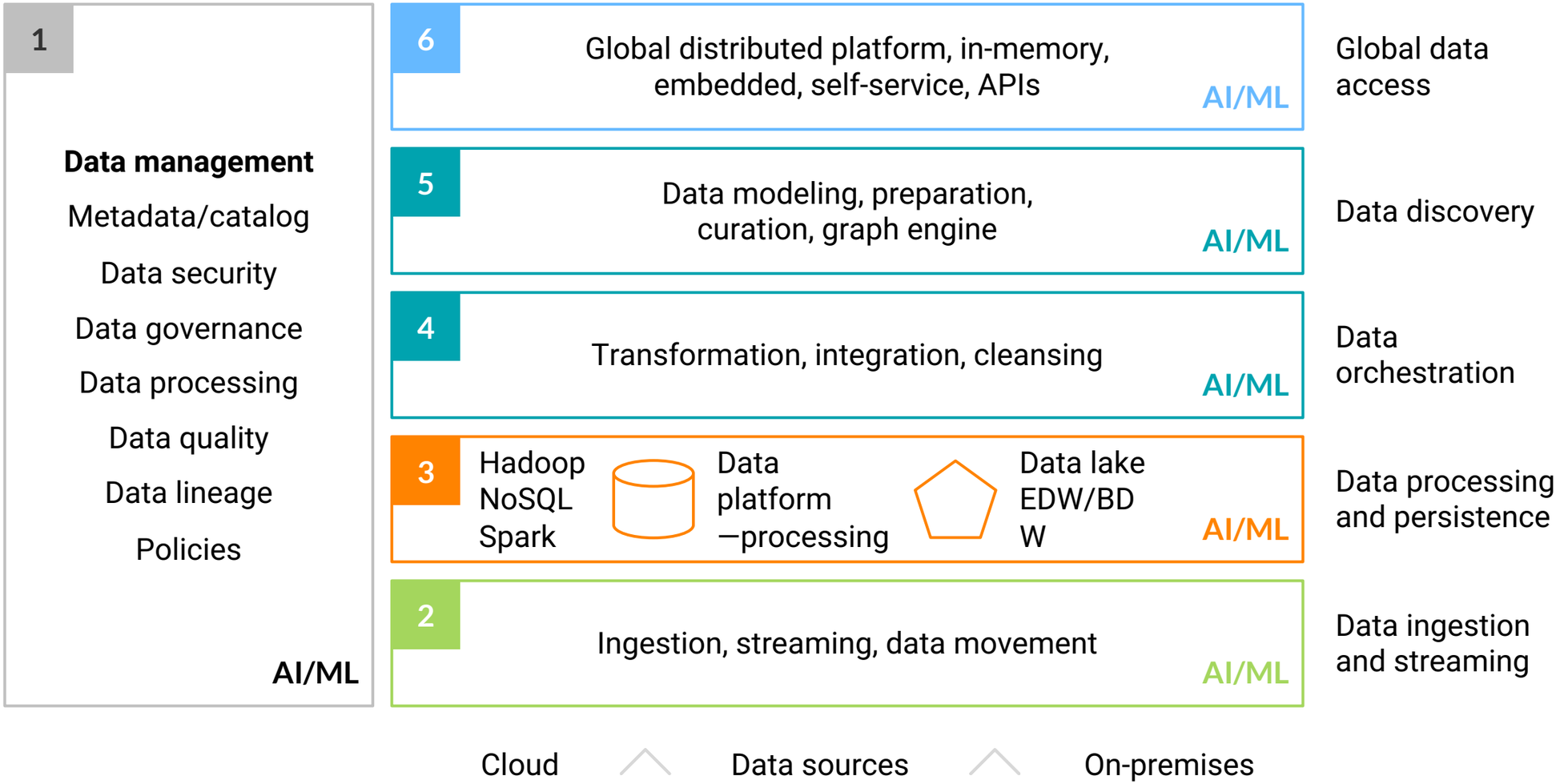


Forrester's Definition

A Data Fabric orchestrates disparate data sources intelligently and securely in a self-service manner, leveraging data platforms such as data lakes, data warehouses, NoSQL, translytical, and others to deliver a unified, trusted, and comprehensive view of customer and business data across the enterprise to support applications and insights.

CDP Hybrid Data Platform – Enabling Data Fabric (2)

Forrester's Definition



CDP Hybrid Data Platform – Enabling Data Lakehouse

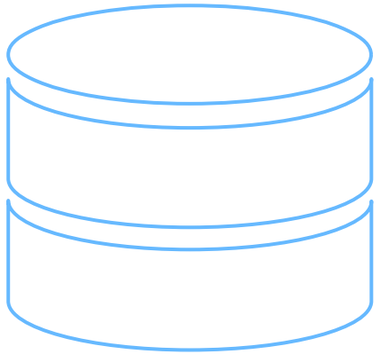
Gartner's Definition

Data lakehouses integrate and unify the capabilities of data warehouses and data lakes, aiming to support AI, BI, ML and data engineering ("multifunction analytics") on a single platform.

The “Lakehouse” – Best of Both Worlds

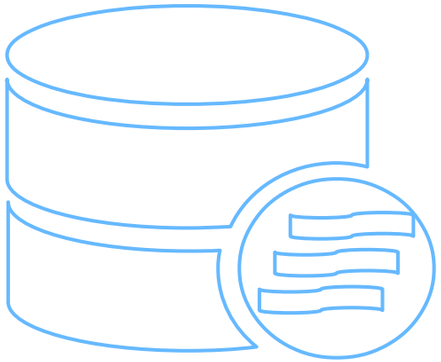
When ALL data needs to come together to answer critical business challenges

DATA WAREHOUSE



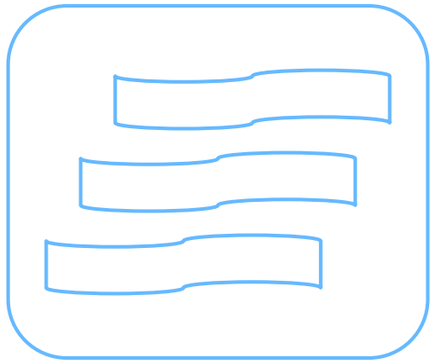
HIGHLY CURATED
PRECISE
COMPLETE
WELL GOVERNED

LAKEHOUSE



EXPLORE AND DISCOVER
TRENDS AND STATISTICS
PREDICT AND LEARN
RAPID TIME TO INSIGHT

DATA LAKE



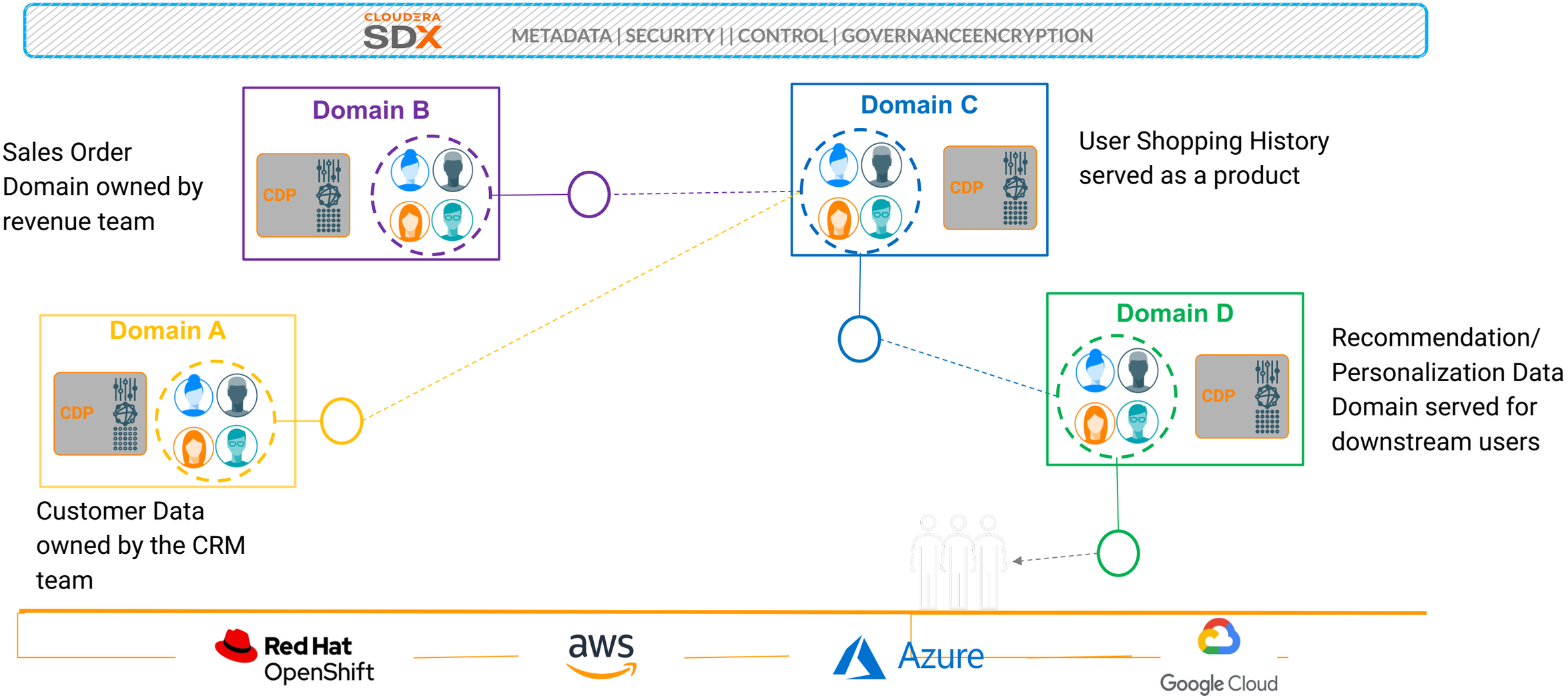
VARYING CURATION
DEGREES OF ACCURACY
SEMI, UNSTRUCTURED
HIGH GRANULARITY

CDP Hybrid Data Platform – Enabling Data Mesh

Data as a Product

Distributed data products oriented around domains and owned by independent cross-functional teams who have embedded data engineers and data product owners, using common data infrastructure as a platform to host, prep and serve their data assets.

CDP Hybrid Data Platform – Enabling Data Mesh (2)



Chapter Topics

Introduction

- About This Course
- Introductions
- About Cloudera
- **About Cloudera Educational Services**
- Course Logistics

Cloudera Educational Services

We offer a variety of ways to take our courses

- Instructor-led, both in physical and virtual classrooms
- Private and customized courses also available
- Self-paced, through Cloudera OnDemand

Courses for all kinds of data professionals

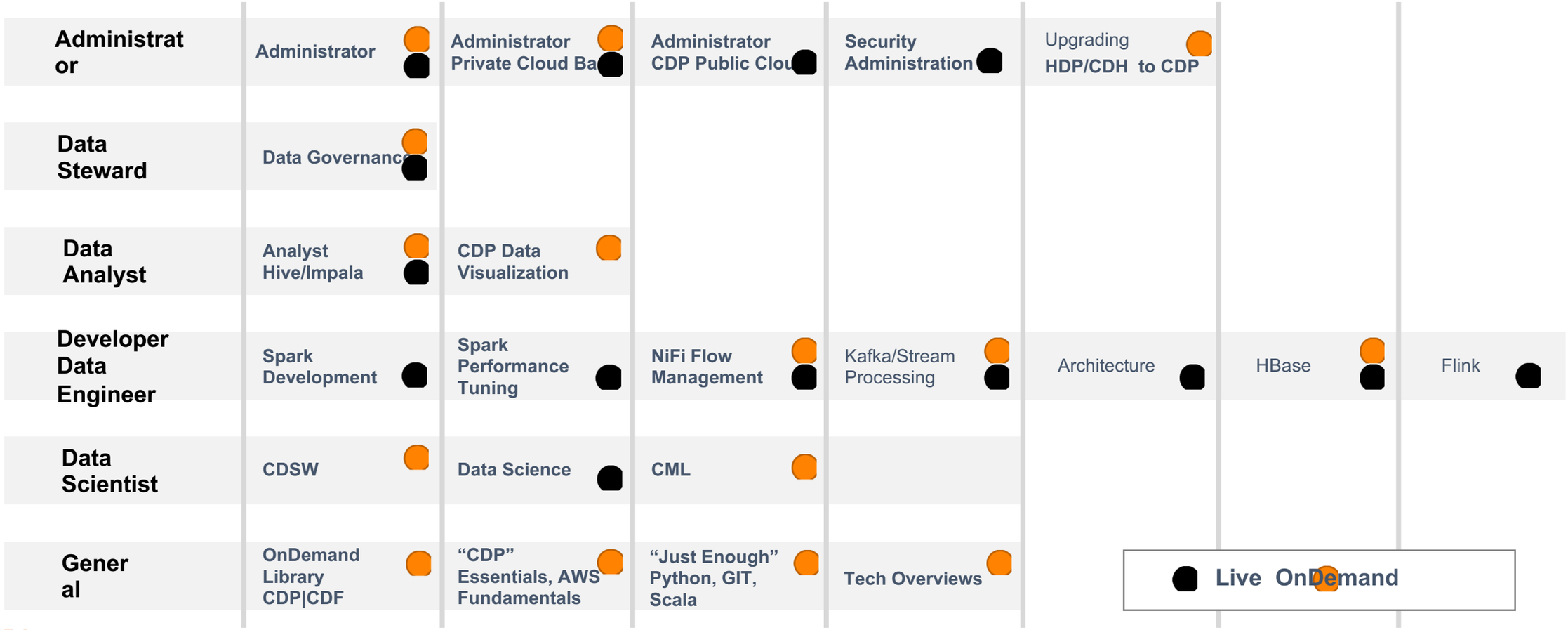
- Executives and managers
- Data scientists and machine learning specialists
- Data analysts
- Developers and data engineers
- System administrators
- Security professionals

Cloudera Education Catalog

A broad portfolio across multiple platforms

- Not all courses shown here

See our website for the complete catalog



Cloudera OnDemand

Our OnDemand catalog includes

- Courses for developers, data analysts, administrators, and data scientists
- Exclusive OnDemand-only courses, such as those covering CDP Public Cloud Administration and Cloudera Data Science Workbench
- Free courses

Features include

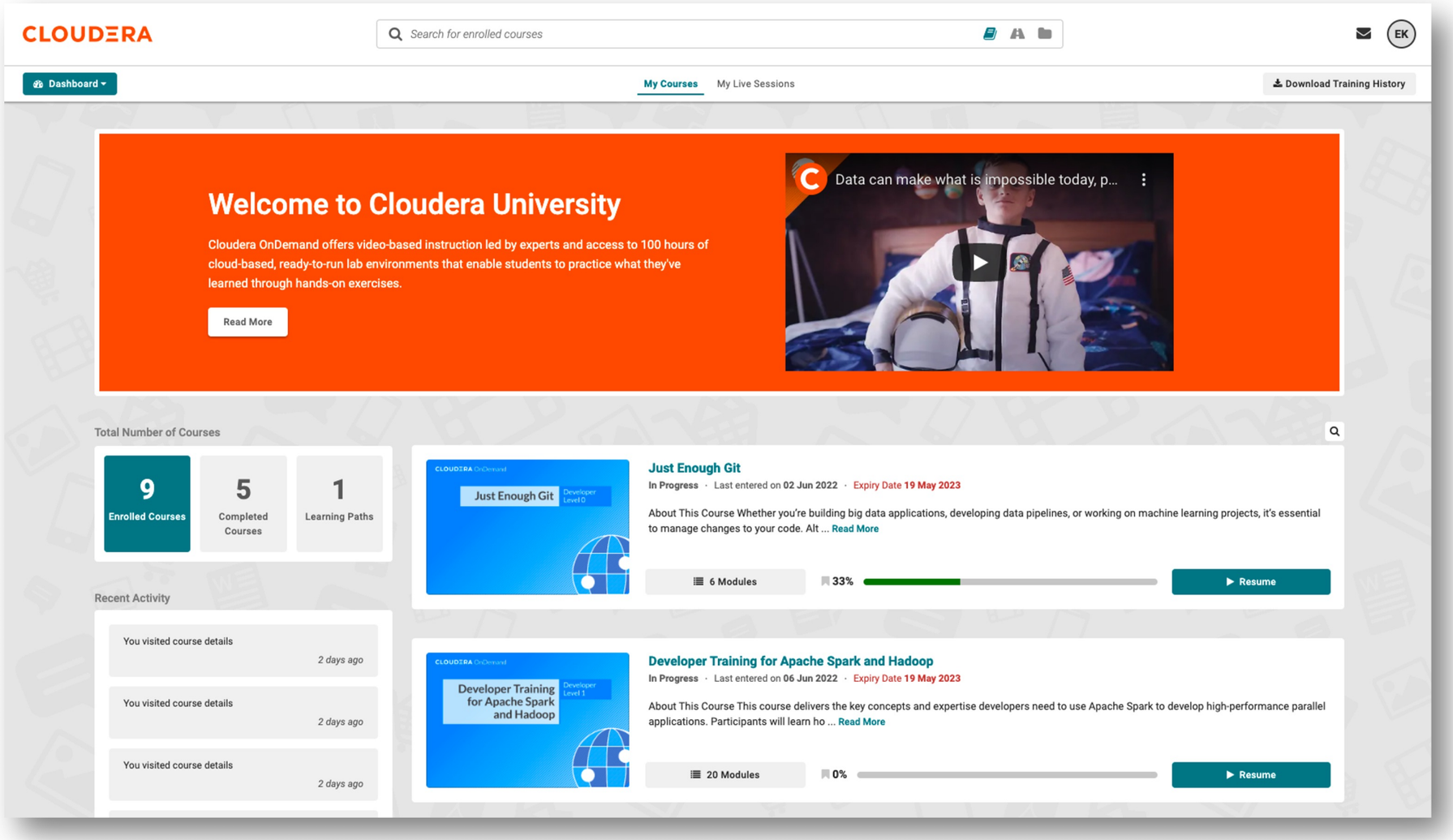
- Video lectures and demonstrations
- Hands-on exercises through a browser-based virtual environment

Purchase access to a library of courses or individual courses

- Full OnDemand Library subscription

Accessing Cloudera OnDemand

Cloudera OnDemand subscribers can access their courses online through a web browser



CDP Certification Program

New role-based certifications

- CDP Certified Generalist
- CDP Certified Administrator
  - Private Cloud
  - Public Cloud
- CDP Certified Developer
- CDP Certified Analyst

Convenient online, proctored exams

Digital badges

www.cloudera.com/about/training/cdp-certification.html

Chapter Topics

Introduction

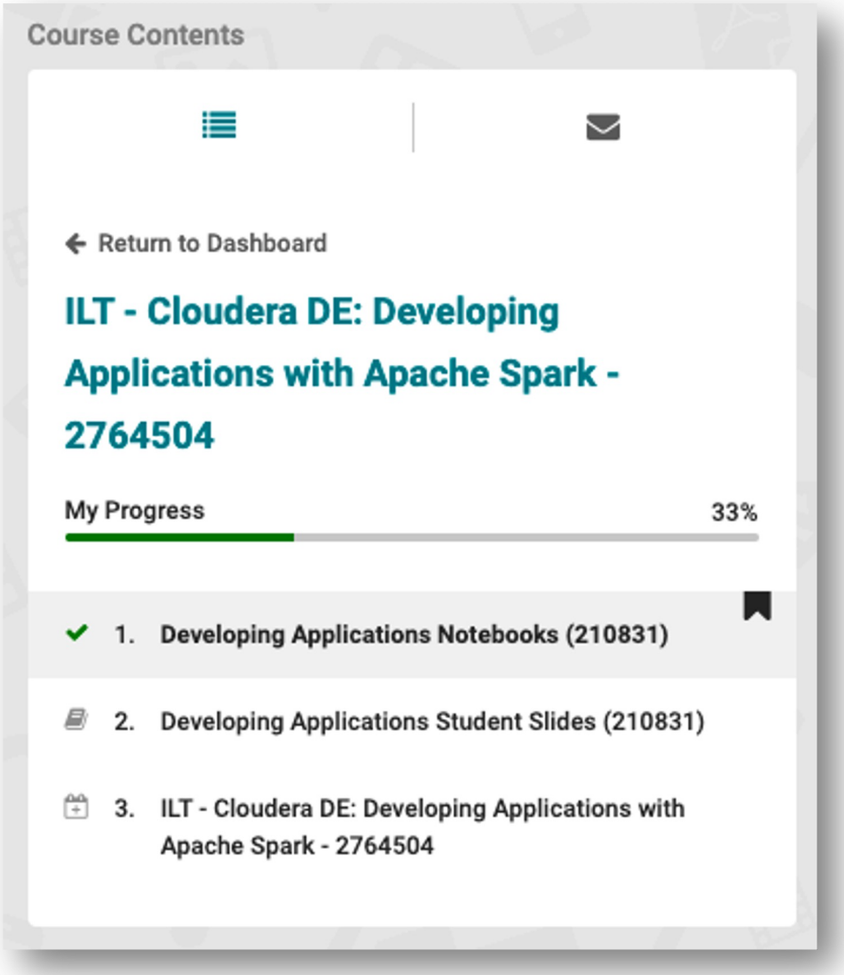
- About This Course
- Introductions
- About Cloudera
- About Cloudera Educational Services
- **Course Logistics**

Logistics

- Class start and finish time
- Lunch
- Breaks
- Restrooms
- Wi-Fi access
- Exercise Environment

Downloading the Course Materials

1. **Log in using https://education.cloudera.com/**
   - Click **Sign In** on the top right
     - If necessary, create an account
     - If you have forgotten your password, use the **Forgot your password** link
2. **Locate the course**
   - Find the course in your list of enrollments, or enter the course title in the search bar
   - Click on the course title
3. **Access the materials**
   - Click on **Content** across the top
   - Click on the module to view or download the contents
   - If there is more than one module, use the Course Contents panel on the left to navigate



CLOUDERA

Why Data Engineering?

Chapter 2

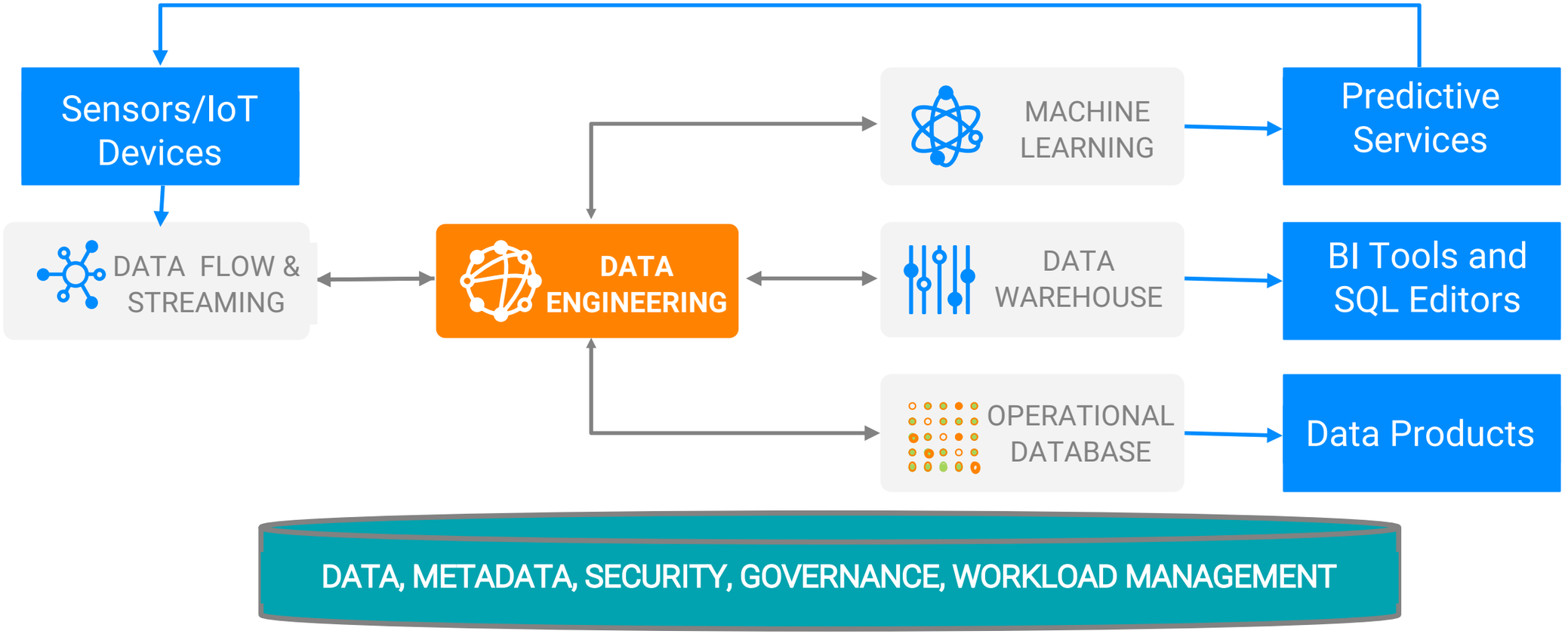
Course Chapters

- Introduction
- **Why Data Engineering?**
- Introduction to Zeppelin
- HDFS Introduction
- YARN Introduction
- Distributed Processing History
- Working With RDDs
- Working With Data Frames
- Introduction to Apache Hive
- Transforming Data with Hive
- Data Engineering with Hive
- Hive Spark Integration
- Distributed Processing Challenges
- Spark Distributed Processing
- Spark Distributed Persistence
- Working with the Data Engineering Service
- Working with Airflow
- Workload Manager Introduction
- Conclusion
- Appendix: Working with Datasets in Scala

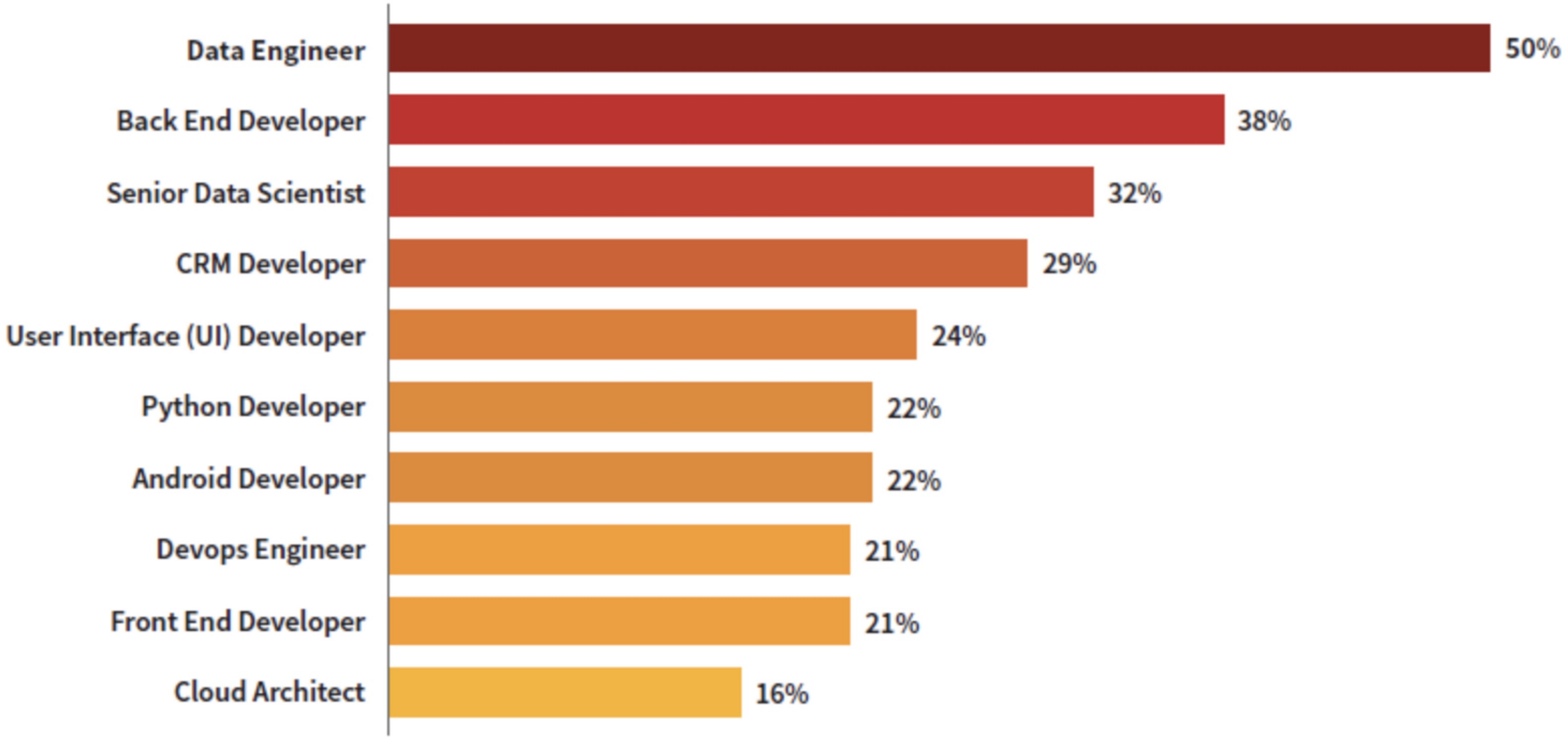
Well defined and managed datasets are the backbone of Machine Learning, Data Warehousing, and Data Cataloging

Data Engineering is Central

To Powering Business Analytics, ML, and Data Products

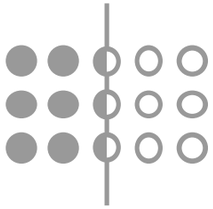


The Fastest Growing, In-Demand Need for Enterprise Businesses



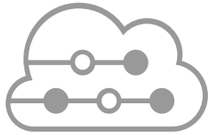
The Challenges with Data Engineering

Managing Spark Resources



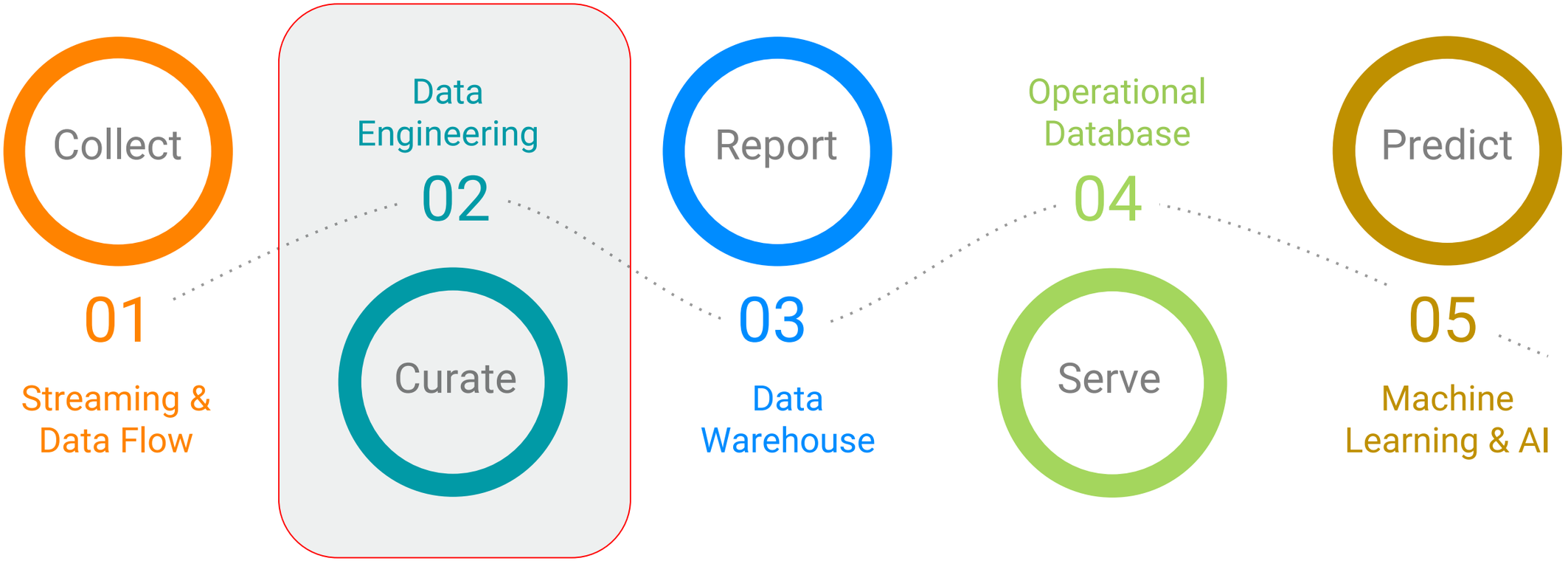
Orchestrating Complex Pipelines

Visibility & Troubleshooting



Secure & Fast Delivery

Data Engineering within the Data Lifecycle



POWERED BY CLOUDERA SDX

Security | Governance | Lineage | Management | Automation

An Integrated, Purpose-Built Experience for Data Engineers

OPTIMIZED FOR DATA ENGINEERS

Streamlined service for scheduling, monitoring, debugging, and promoting data pipelines quickly & securely.

EVERYTHING YOU NEED TO POWER ANALYTICS

- Inherited governance
- Deliver data pipelines to CDW, CML, or COD easily
- Portable and flexible

COMPLETE DATA PIPELINE MANAGEMENT

- Monitoring & alerting for catching issues early
- Visual troubleshooting
- Governed and secure workflows with SDX

An Integrated, Purpose-Built Experience for Data Engineers

CONTAINERIZED, MANAGED SPARK SERVICE

- Autoscaling compute
- Governed & secure with Cloudera SDX
- Mix version deployments

APACHE AIRFLOW SCHEDULING

- Open preferred tooling
- Orchestrate complex data pipelines
- Manage & schedule dependencies easily

TUNING & VISUAL TROUBLESHOOTING

- Resolve issues fast with real-time visual performance profiling
- Complete monitoring & alerting capabilities

SIMPLIFIED JOB MANAGEMENT & APIS

- Full lifecycle mgmt.
- API-driven pipeline automation for any service
- Any language: SQL, Java, Scala, Python

CLOUDERA

Introduction to Zeppelin

Chapter 3

Course Chapters

- Introduction
- Why Data Engineering
- **Introduction to Zeppelin**
- HDFS Introduction
- YARN Introduction
- Distributed Processing History
- Working With RDDs
- Working With Data Frames
- Introduction to Apache Hive
- Transforming Data with Hive
- Data Engineering with Hive
- Hive Spark Integration
- Distributed Processing Challenges
- Spark Distributed Processing
- Spark Distributed Persistence
- Working with the Data Engineering Service
- Working with Airflow
- Workload Manager Introduction
- Conclusion
- Appendix: Working with Datasets in Scala

Objectives

- **By the end of this chapter, you will be able to:**
  - Understand the motivation for notebooks
  - Understand the structure of a Zeppelin notebook
  - Run and create Zeppelin notebooks using best practices

Chapter Topics

Introduction to Zeppelin

- **Why Notebooks?**
- Zeppelin Notes
- Demo: Apache Spark In 5 Minutes

Required For a Scientific Approach to Data Analysis

- **To be able to perform analysis on data in a scientific manner, we need a tool that allows us to perform reproducible data analysis in which executable code is interspersed with paragraphs of text and visualizations that document our train of thoughts (literate programming)**
- **Several tools meet those requirements, here are the most popular ones:**
  - RStudio 
  - Jupyter 
  - Zeppelin 
- **All of them can be used as IDEs with our Cloudera Data Science Workbench tool**

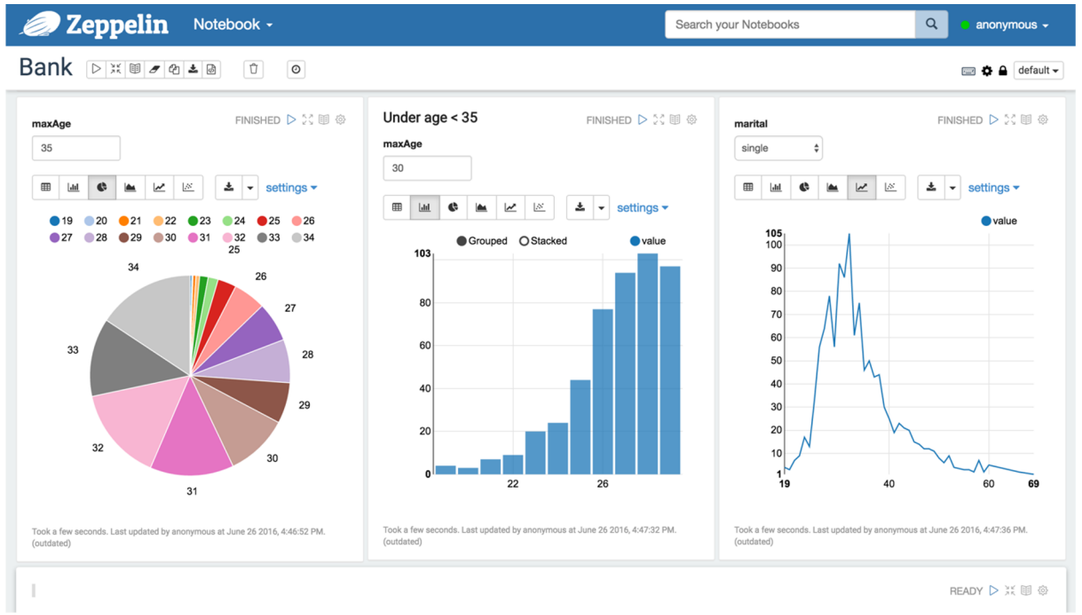
Chapter Topics

Introduction to Zeppelin

- Why Notebooks?
- **Zeppelin Notes**
- Demo: Apache Spark In 5 Minutes

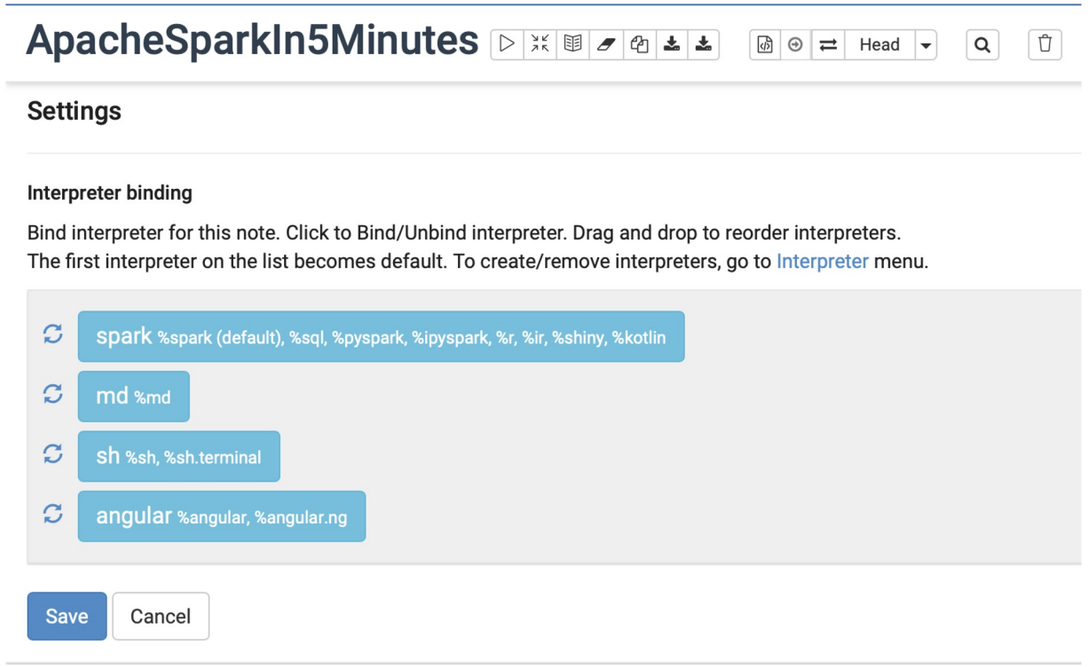
What is Apache Zeppelin?

- **An open source project from the Apache Software Foundation**
  - 2013, NFLabs started the Zeppelin project
  - 2014-12-23, the Zeppelin project became incubation project in Apache Software Foundation
  - 2016-06-18, the Zeppelin project graduated incubation and became a Top Level Project in Apache Software Foundation
- **A multi-purpose notebook system for**
  - Data ingestion
  - Data discovery
  - Data analytics
  - Data visualization and collaboration



Anatomy of a Note

- A Zeppelin note is a sequence of paragraphs
- Each paragraph is bound to an interpreter
- Each note has a list of available interpreters which is a subset of all the interpreters installed on the Zeppelin server
- The first element of the list is the default interpreter for the note
- Each paragraph should start by specifying the interpreter to which it is bound unless it is the default

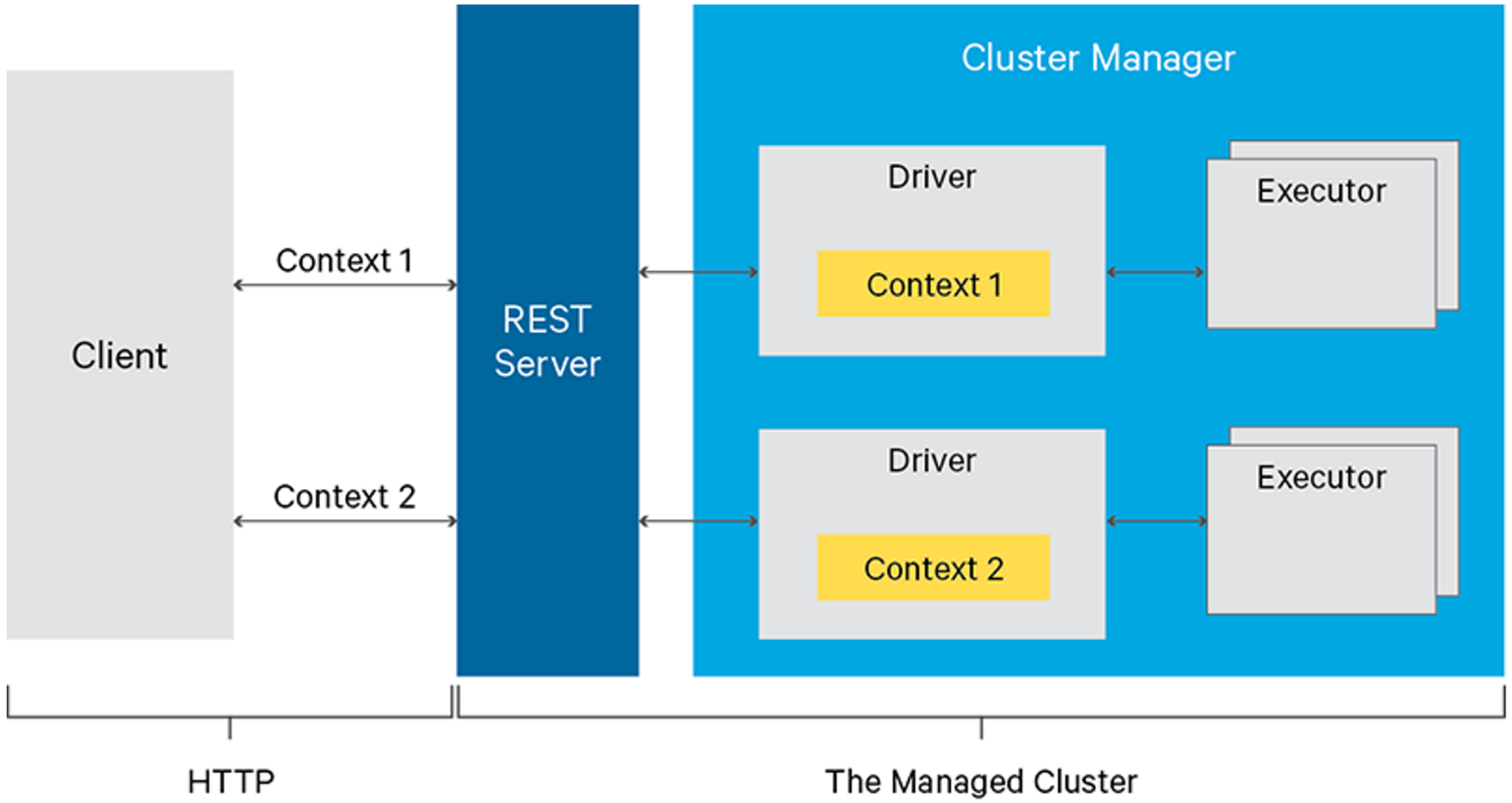


Available Interpreters

- **The latest version of Zeppelin contains over 30 interpreters**
- **In CDP DC the following interpreters are available by default:**
  - angular
  - livy
  - md
- **The default one is livy**
- **This is a direct consequence of the secured design of CDP DC**
- **If you need an additional interpreter check with your favourite admin whether it can be installed securely**

What is Apache Livy?

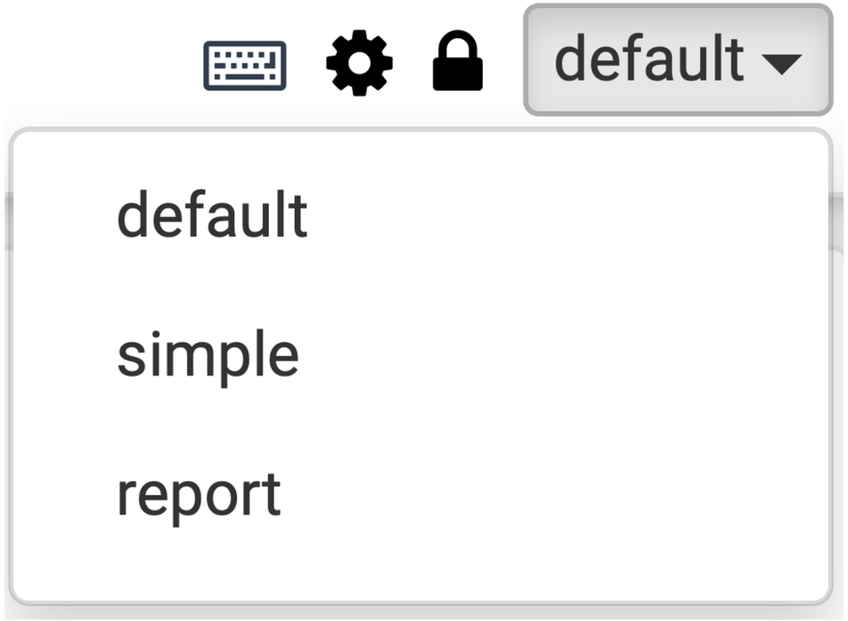
- Livy enables multitenant and secure communication with a Spark cluster over a REST interface



Note Formatting

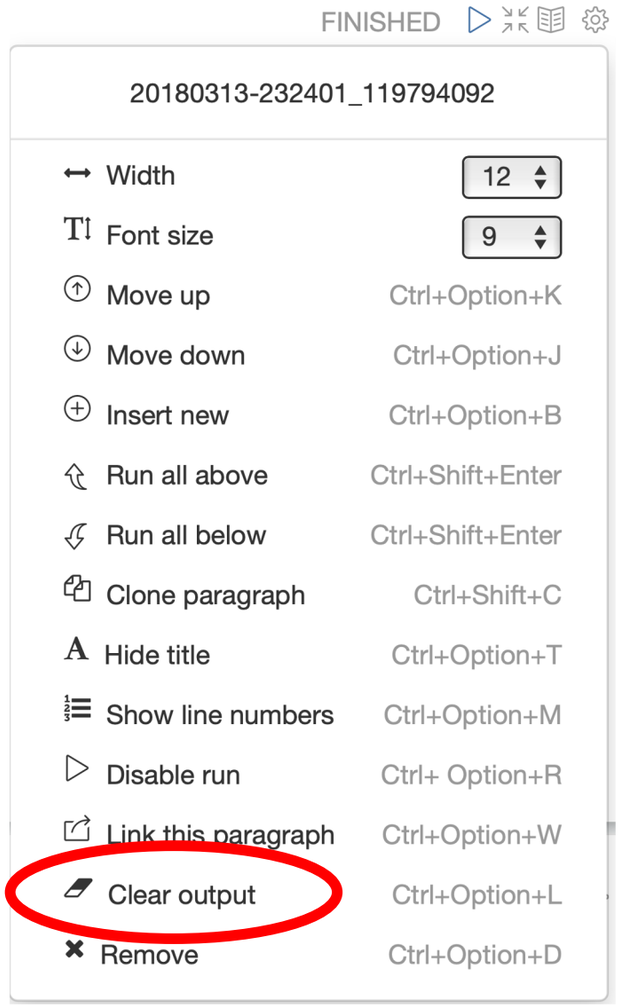
- **Note owners can control all paragraphs at the note level, including:**
  - Hide/Show all code
  - Hide/Show all output
  - Clear all output

- **There are also two additional note views**
  - Simple: Removes note-level controls
  - Report: Removes note-level controls and all code



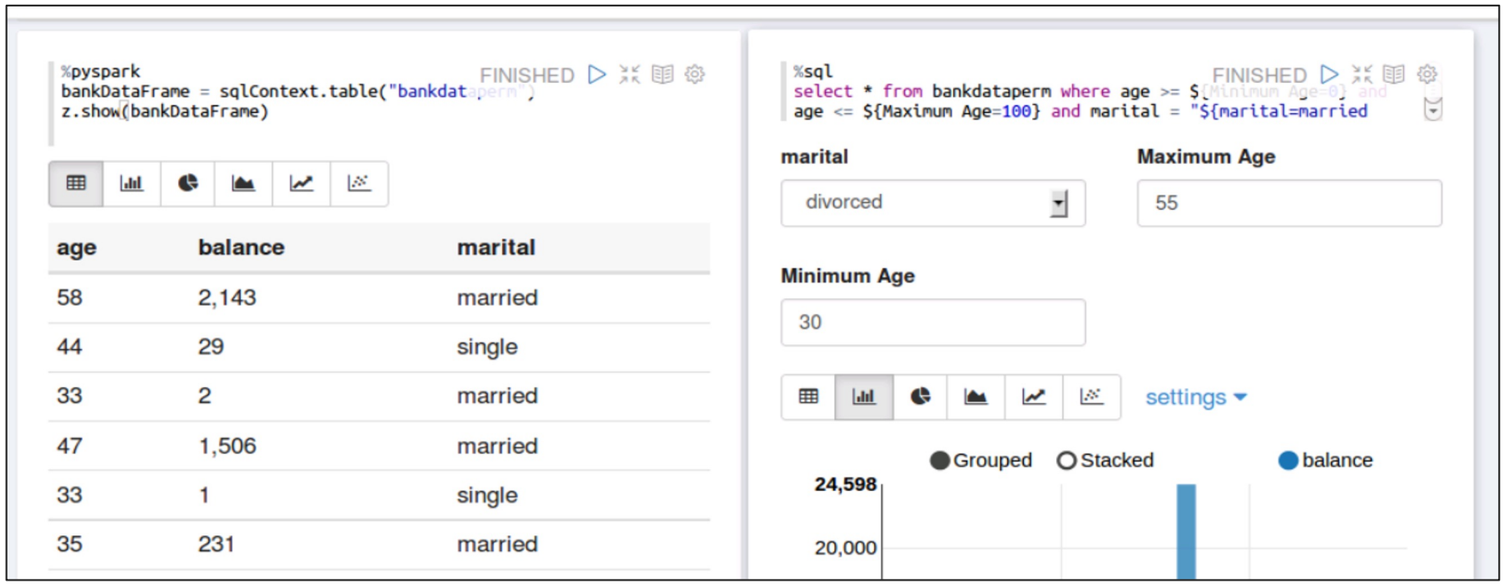
Paragraph Formatting

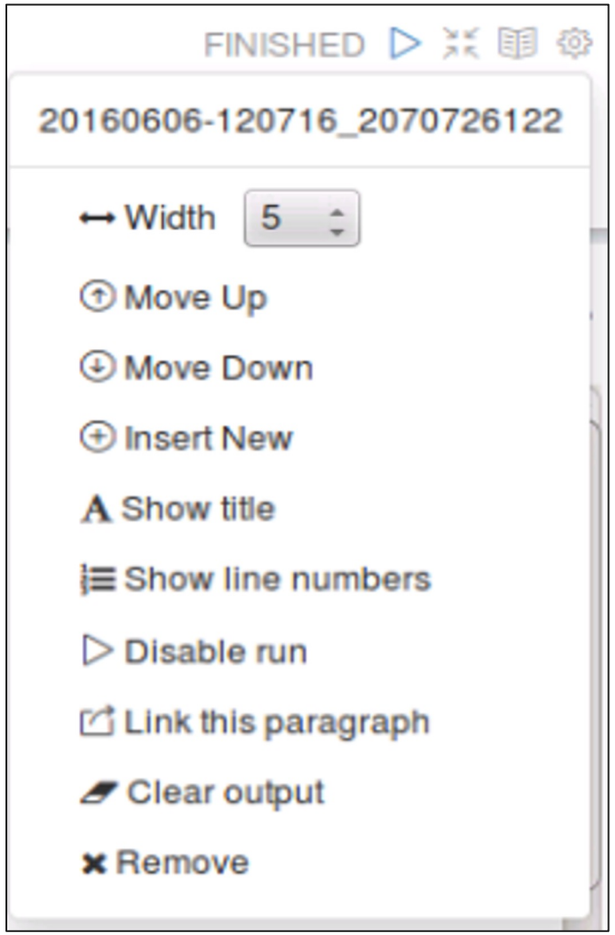
- **Paragraphs also contain formatting settings, including:**
  - Hide/Show paragraph code
  - Hide/Show paragraph output
  - Clear paragraph output is available in the settings menu (gear icon)



Paragraph Enhancement - Width

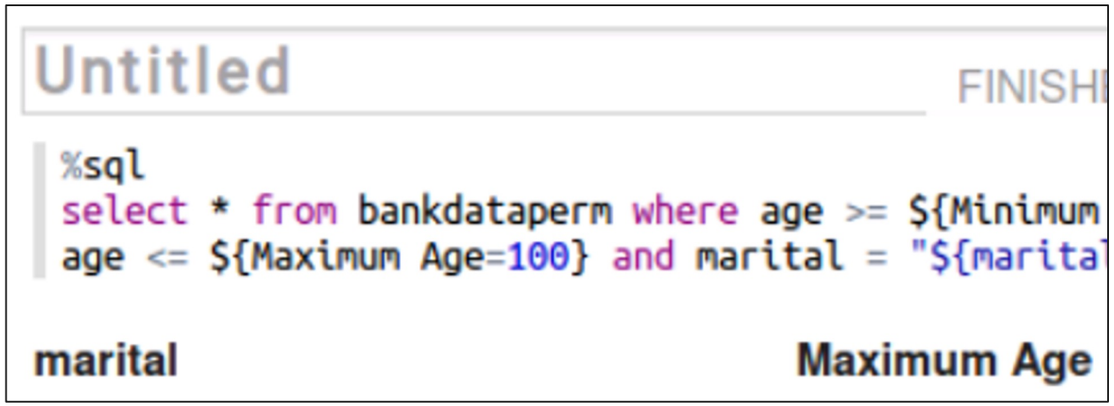
- **Width: Controls width of the paragraph in the note, allowing multiple paragraphs to be displayed in a row**

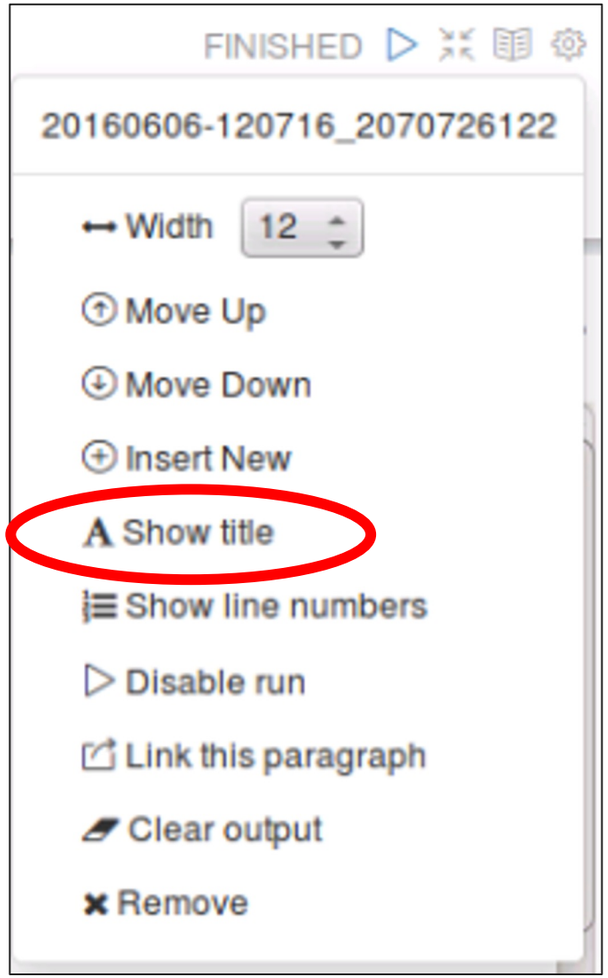




Paragraph Enhancement - Show Title

- **Paragraph titles can be added for clarity**



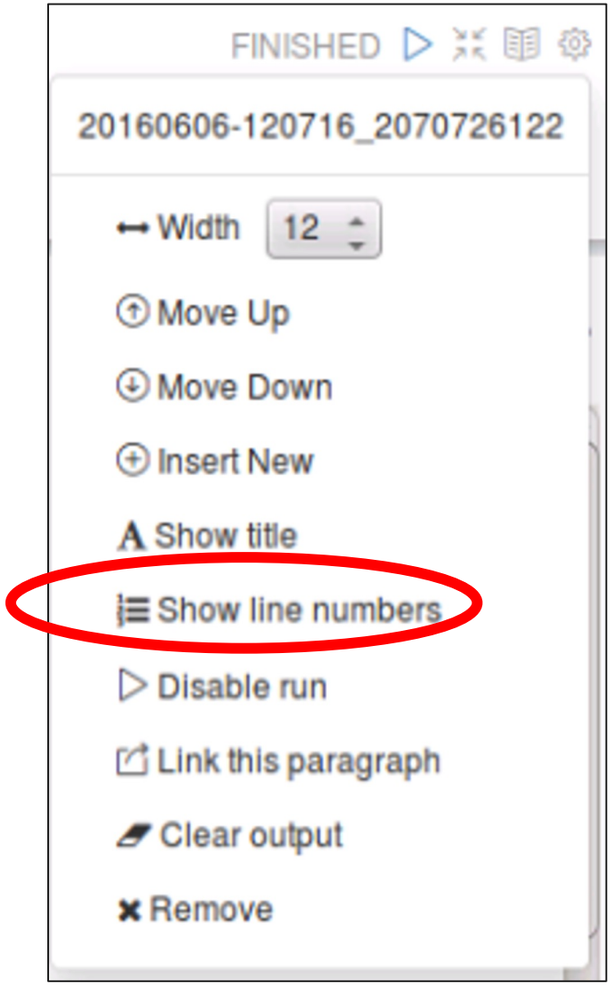


Paragraph Enhancement - Line Numbers

- **Line numbers can be added to paragraph code**

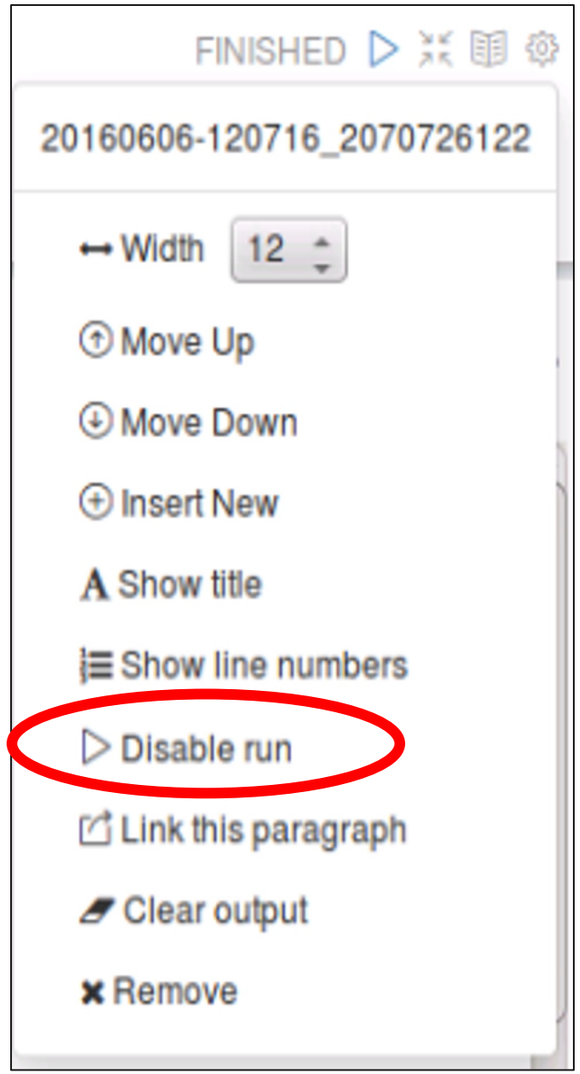
```
1 %sql
2 select * from bankdataperm where age
  and age <= ${Maximum Age=100} and marital =
```

marital **Maximum Age**



Disable Paragraph Output Changes

- **Disable the paragraph run feature to lock the output of a paragraph**
- **Changes to Dynamic Forms or code will not be reflected in the paragraph**



Best Practices

- **Save your notes outside of the Zeppelin home folder**
- **Clear the outputs of your notes before exporting them**
  - Saves disk space
  - Reduces the risk of not being able to import them back in
- **Minimize the dependencies of your notes with your local environment to increase reproducibility**
  - Download your data from cloud storage if possible
- **Make your note error free when run twice**
  - Make the extra effort to avoid triggering an error if the same paragraph runs twice
- **Disable your markdown paragraphs and hide their codes**
  - Saves CPU cycles and looks cleaner
- **Give your paragraphs titles**
- **Make good use of markdown paragraphs and the visualization features to tell your story**

Chapter Topics

Introduction to Zeppelin

- Why Notebooks?
- Zeppelin Notes
- **Demo: Apache Spark In 5 Minutes**

CLOUDERA

Working with DataFrames

Chapter 8

Course Chapters

- Introduction
- Why Data Engineering?
- Introduction to Zeppelin
- HDFS Introduction
- YARN Introduction
- Distributed Processing History
- Working With RDDs
- **Working With Data Frames**
- Introduction to Apache Hive
- Transforming Data with Hive
- Data Engineering with Hive
- Hive Spark Integration
- Distributed Processing Challenges
- Spark Distributed Processing
- Spark Distributed Persistence
- Working with the Data Engineering Service
- Working with Airflow
- Workload Manager Introduction
- Conclusion
- Appendix: Working with Datasets in Scala

Lesson Objectives

By the end of this chapter, you will be able to:

- **Describe how Spark SQL fits into the Spark stack**
- **Start and use the Python and Scala Spark interpreters**
- **Create DataFrames and perform simple queries**

Chapter Topics

Working with DataFrames

- **Introduction to DataFrames**
- Exercises
  - Introducing DataFrames
  - Reading and Writing DataFrames
  - Working with Columns
  - Working with Complex Types
  - Combining and Splitting DataFrames
  - Summarizing and Grouping DataFrames
  - Working with UDFs
  - Working with Windows

RDDs Limitations

- **Low level API**
  - You specify details (the how) not intent (the what)
  - API has little intelligence for dealing with common data formats
    - e.g. JSON
- **Opaque to Spark engine (uses arbitrary lambdas)**
  - Queries can't easily be optimized by Spark
  - Not hard to write inefficient transformations
  - Often not obvious
  - And Spark can't make them better
- **No support for SQL-like querying**
  - Limits applicability
  - SQL is well known

DataFrames and Datasets

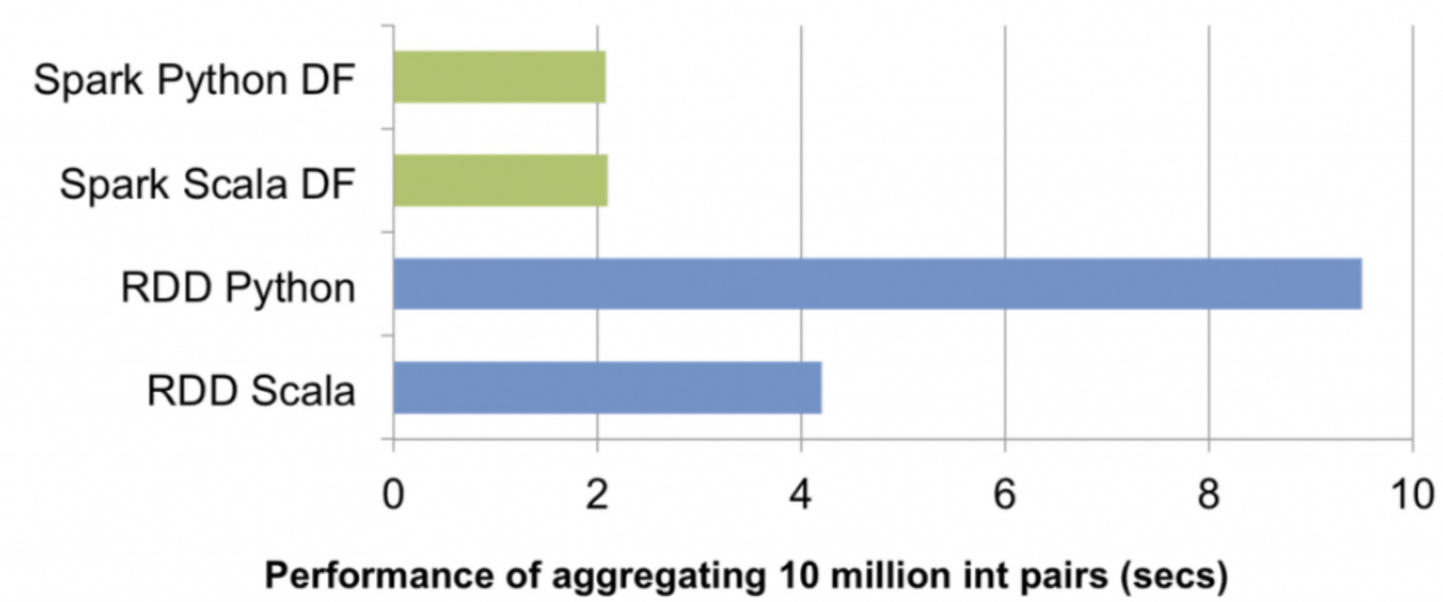
- **DataFrames and Datasets are the primary representation of data in Spark.**
- **DataFrames represent structured data in a tabular form.**
  - DataFrames model data similar to tables in an RDBMS.
  - DataFrames consist of a collection of loosely typed Row objects.
  - Rows are organized into columns described by a schema.
- **Datasets represent data as a collection of objects of a specified type.**
  - Datasets are strongly-typed—type checking is enforced at compile time rather than run time.
  - An associated schema maps object properties to a table-like structure of rows and columns.
  - Datasets are only defined in Scala and Java.
  - DataFrame is an alias for Dataset[Row]—Datasets containing Row objects.

DataFrames and Datasets: Some History

- **Introduced as SchemaRDD (1.0), renamed to DataFrame (1.3)**
- **Dataset type introduced in Spark 1.6**
  - Separate type from DataFrame for backwards compatibility
- **Dataset and DataFrame unified in 2.0**
  - DataFrame is now a typedef for Dataset[Row] (Scala)
  - The API is defined in Dataset, and divided into sections
    - Untyped operations derive historically from DataFrame
    - Typed operations derive historically from Dataset
- **A bit confusing — especially if you worked with early releases**
- **We'll use DataFrame to refer to the untyped API**

Why DataFrames (and Datasets)? - More efficient code

- DataFrames based code is translated to RDD code through a process that involves a sequence of optimisers (Catalyst and Tungsten) that produce the best RDD code possible
- Catalyst understands the structure of data & semantics of operations and performs optimizations accordingly
- The Tungsten storage format is 4 times more efficient than its Java counterpart



Why DataFrames (and Datasets)? - Cleaner Code

- **The syntax is easier on the eyes: these three snippets of code perform the same processing**

```
> val folksRDD=sc.textFile("people.txt")                    // Get the data
> folksRDD.map(_.split(" "))                                // Split it into fields
   .map(x => (x(1), Array(x(2).toDouble, 1)))               // Pair up the data
   .reduceByKey( (x,y) => Array(x(0)+y(0), x(1)+y(1)) )     // Count
   .map(x => Array(x._1, x._2(0)/x._2(1)))                  // Figure average
   .collect                                                 // Get results
```

Language: *Scala*

```
> val folksDF=spark.read.json("people.json") // Get the data
> folksDF.groupBy($"gender")                 // Group by gender
   .agg(avg($"age"))                         // Get average age of groups
   .show                                     // Display data
```

Language: *Scala*

```
> val folksDF=spark.read.json("people.json") // Get the data
> folksDF.createOrReplaceTempView("people")  // Setup for using SQL
> spark.sql("SELECT gender, avg(age) FROM people GROUP BY gender").show
```

Language: *Scala*

DataFrames and Rows

- **DataFrames contain a collection of Row objects**
  - Rows contain an ordered collection of values
  - Row values can be basic types (such as integers, strings, and floats) or collections of those types (such as arrays and lists)
  - A schema maps column names and types to the values in a row

Spark Session

- **The main entry point for the Spark API is a Spark session**
- **The Spark interpreters provide a preconfigured SparkSession object called spark**
- **The SparkSession class provides functions and attributes to access all of Spark functionality**
- **Examples include**
  - sql: execute a Spark SQL query
  - catalog: entrypoint for the Catalog API for managing tables
  - read: function read data from a file or other data source
  - conf: object to manage Spark configuration settings
  - sparkContext: entry point for core Spark API

Example: Creating a DataFrame (1)

- **The users.json text file contains sample data**
  - Each line contains a single JSON record that can include a name, age, and postal code field

```
{"name":"Alice", "pcode":"94304"}
{"name":"Brayden", "age":30, "pcode":"94304"}
{"name":"Carla", "age":19, "pcode":"10036"}
{"name":"Diana", "age":46}
{"name":"Étienne", "pcode":"94104"}
```

Example: Creating a DataFrame (2)

- Create a DataFrame using spark.read
- Returns the Spark session's DataFrameReader
- Call json function to create a new DataFrame

```
> usersDF = spark.read.json("users.json")
```

Language: *Python*

Example: Creating a DataFrame (3)

- DataFrames always have an associated schema
- DataFrameReader can infer the schema from the data
- Use printSchema to show the DataFrame's schema

```
> usersDF = spark.read.json("users.json")
> usersDF.printSchema()
root
|--age: long (nullable = true)
|--name: string (nullable = true)
|--pcode: string (nullable = true)
```

Language: *Python*

Example: Creating a DataFrame (4)

- **The show method displays the first few rows in a tabular format**

```
> usersDF = spark.read.json("users.json")

> usersDF.printSchema()

root
|--age: long (nullable = true)
|--name: string (nullable = true)
|--pcode: string (nullable = true)

> usersDF.show()

+----+-------+-----+
| age|   name|pcode|
+----+-------+-----+
|null|  Alice|94304|
|  30|Brayden|94304|
|  19|  Carla|10036|
|  46|  Diana| null|
|null|Etienne|94104|
+----+-------+-----+
```

Language: *Python*

DataFrame Operations

- **There are two main types of DataFrame operations**
  - Transformations create a new DataFrame based on existing one(s)
    - Transformations are executed in parallel by the application's executors
  - Actions output data values from the DataFrame
    - Output is typically returned from the executors to the main Spark program (called the driver) or saved to a file

DataFrame Operations: Actions

- **Some common DataFrame actions include**
  - count: returns the number of rows
  - first: returns the first row (synonym for head())
  - take(n): returns the first n rows as an array (synonym for head(n))
  - show(n): display the first n rows in tabular form (default is 20 rows)
  - collect: returns all the rows in the DataFrame as an array
  - write: save the data to a file or other data source

Example: take Action

```
> usersDF = spark.read.json("users.json")

> users = usersDF.take(3)
  [Row(age=None, name=u'Alice', pcode=u'94304'),
   Row(age=30, name=u'Brayden', pcode=u'94304'),
   Row(age=19, name=u'Carla', pcode=u'10036')]
```

Language: *Python*

```
> val usersDF = spark.read.json("users.json")

> val users = usersDF.take(3)
  usersDF: Array[org.apache.spark.sql.Row] =
  Array([null,Alice,94304],
        [30,Brayden,94304],
        [19,Carla,10036])
```

Language: *Scala*

DataFrame Operations: Transformations (1)

- **Transformations create a new DataFrame based on an existing one**
  - The new DataFrame may have the same schema or a different one
- **Transformations do not return any values or data to the driver**
  - Data remains distributed across the application's executors
- **DataFrames are immutable**
  - Data in a DataFrame is never modified
  - Use transformations to create a new DataFrame with the data you need

DataFrame Operations: Transformations (2)

- **Some common DataFrame transformations include**
  - select: only the specified columns are included
  - where: only rows where the specified expression is true are included (synonym for filter)
  - orderBy: rows are sorted by the specified column(s) (synonym for sort)
  - join: joins two DataFrames on the specified column(s)
  - limit(n): creates a new DataFrame with only the first n rows
  - collect: returns all the rows in the DataFrame as an array
  - write: save the data to a file or other data source

Example: select and where Transformations

```
> nameAgeDF = usersDF.select("name","age")
> nameAgeDF.show()
+-------+----+
|   name| age|
+-------+----+
|  Alice|null|
|Brayden|  30|
|  Carla|  19|
|  Diana|  46|
|Etienne|null|
+-------+----+
> over20DF = usersDF.where("age > 20")
> over20DF.show()
+---+-------+-----+
|age|   name|pcode|
+---+-------+-----+
| 30|Brayden|94304|
| 46|  Diana| null|
+---+-------+-----+
```

Language: *Python*

Defining Queries

- **A sequence of transformations followed by an action is a *query***

```
> nameAgeDF = usersDF.select("name","age")
> nameAgeOver20DF = nameAgeDF.where("age > 20")
> nameAgeOver20DF.show()

+---+-------+
|age|   name|
+---+-------+
| 30|Brayden|
| 46|  Diana|
+---+-------+
```

Language: *Python*

Chaining Transformations (1)

- **Transformations in a query can be chained together**
- **These two examples perform the same query in the same way**
  - Differences are only syntactic

```
> nameAgeDF = usersDF.select("name","age")
> nameAgeOver20DF = nameAgeDF.where("age > 20")
> nameAgeOver20DF.show()
```

Language: *Python*

```
> nameAgeDF = usersDF.select("name","age").where("age > 20").show()
```

Language: *Python*

Chaining Transformations (2)

- **This is the same example with Scala**
  - The two code snippets are equivalent

```
> val nameAgeDF = usersDF.select("name","age")
> val nameAgeOver20DF = nameAgeDF.where("age > 20")
> nameAgeOver20DF.show
```

Language: *Scala*

```
> val nameAgeDF = usersDF.select("name","age").where("age > 20").show
```

Language: *Scala*

Knowledge Check

1. What is Spark SQL, and why do we use it?

2. How do you read data using the SparkSession? What kinds of data can be read?

3. What is a DataFrame? A Dataset?

4. True or False? - DataFrames are convenient but will never outperform native RDD based code.

Essential Points

- **DataFrames create another abstraction between the developers and data**
- **DataFrames have built in optimizers and outperforms core spark in speed**
- **DataFrames represent structured data in tabular form by applying a schema Types of DataFrame operations**
  - Transformations create new DataFrames by transforming data in existing ones
  - Actions collect values in a DataFrame and either save them or return them to the Spark driver
- **A query consists of a sequence of transformations followed by an action**

Chapter Topics

Working with DataFrames

- Introduction to DataFrames
- **Exercises**
  - Introducing DataFrames
  - Reading and Writing DataFrames
  - Working with Columns
  - Working with Complex Types
  - Combining and Splitting DataFrames
  - Summarizing and Grouping DataFrames
  - Working with UDFs
  - Working with Windows