ChatGPT

Apache Spark on AWS – 20-Day Data Engineering Learning Plan

Step 1: Knowledge Assessment – Skill Tree & Core Concepts

Core Components of Apache Spark: Apache Spark consists of several key modules. At its heart is **Spark Core**, which provides the basic engine for distributed data processing (resilient distributed datasets, task scheduling, memory management, fault recovery, etc.) [1] [2]. Built on Spark Core are higher-level libraries:
- **Spark SQL and DataFrames** – for structured data processing using SQL queries or DataFrame API [3]. This is a primary interface for data engineers, allowing schema-based operations and SQL integration (leveraging the Catalyst optimizer for performance) [4] [5].
- **Spark Streaming (Structured Streaming)** – for real-time or micro-batch stream processing of data streams (advanced use, extends Spark Core for processing incoming data in near real-time) [6].
- **Spark MLlib** – Spark's machine learning library, offering scalable algorithms (advanced topic, relevant if your role touches ML).
- **GraphX** – Spark's graph computation library (optional/less common for data engineering).

Spark's **architecture** underpins these components. Key elements include the **Spark Driver** (central coordinator that builds the logical *plan* and schedules tasks), **Executors** (workers that run tasks on partitions of data), and a **Cluster Manager** (e.g. YARN, Kubernetes, or standalone/EMR/Glue in AWS) that allocates resources across nodes [7] [8]. Understanding this helps in tuning and debugging Spark jobs. Spark uses a **lazy evaluation** model: transformations (e.g. `map`, `filter`) build up a **DAG (Directed Acyclic Graph)** of operations without executing immediately [9] [10], and only when an **action** (e.g. `count()`, `collect()`) is called does the execution actually trigger [11]. This design enables Spark to optimize the overall plan for efficiency. Key foundational concepts include **RDDs** (low-level immutable data collections partitioned across the cluster) and higher-level DataFrames/Datasets which provide schema and optimize operations. A solid grasp of **transformations vs. actions, partitioning of data, and Spark's execution model** is critical for effective use of Spark.

Complexity & Dependencies: For a beginner, the Spark SQL/DataFrame API will be more approachable than low-level RDD manipulation. Start with DataFrames and SQL (easier to learn, closer to your SQL background), then later peek into RDDs to understand the fundamentals under the hood [12]. Spark Streaming, MLlib, and GraphX are more advanced; they can be considered **optional** for foundational learning given the 20-day timeframe – focus on them only if time permits or if the job demands those specifically. Spark's internal optimization (Catalyst, Tungsten) and performance tuning (e.g. avoiding shuffles, caching, partition sizing) are intermediate-to-advanced topics; be aware of them but you can postpone deep dives until after mastering the basics.

Prerequisite Knowledge: Since you come from a BI background, you likely have strong SQL skills and data modeling understanding – this will help greatly with Spark SQL and understanding data schemas. You should be comfortable with **Python programming** (because PySpark is the target) – specifically, know how to write scripts, use data structures, and handle data in Python. If your Python is rusty, spend a short time refreshing basics (reading files, functions, simple data manipulations). Also, ensure familiarity with **SQL** for querying data, as Spark's DataFrame API can use SQL queries. Basic understanding of **parallel vs. distributed computing** concepts is important: Spark is a distributed framework, so knowing why we need distributed processing (vs. single-node) and how MapReduce-like paradigms work will give you context [13]. For example, know that MapReduce (Hadoop) was an earlier paradigm and Spark improves on it by in-memory processing and a richer API. Recognize common big data challenges like the "small files problem" or data skew – you don't need deep expertise yet, but awareness helps [13].

AWS Ecosystem Knowledge: Because your goal is **PySpark on AWS**, the learning path must include AWS data engineering services and how Spark integrates with them [14] [15]. Key AWS components to know (and their role in a Spark-based pipeline):
- **Amazon S3:** AWS's scalable object storage, often serving as a **data lake** to store raw and processed data. Spark can read from and write to S3 (it treats S3 like a file system). S3 basics (buckets, keys, file formats) are straightforward (low complexity) but crucial – you'll use S3 to hold input data and Spark output.
- **AWS Glue:** A fully managed, serverless ETL service that runs Spark under the hood. Glue simplifies running Spark jobs without managing clusters [16]. Core Glue elements: *Glue Jobs* (your PySpark code running on a managed Spark cluster), *Glue Crawlers* (automated schema inference that creates metadata in the Glue Data Catalog), and the *Glue Data Catalog* itself (central metastore for table definitions that services like Athena and Glue can use). Glue is moderate complexity: it abstracts cluster management, but you need to learn how to write Glue-compatible PySpark scripts, configure jobs, and handle IAM roles for permissions.
- **AWS IAM:** Identity and Access Management, used to define roles and policies so that your Glue jobs (Spark) can securely access S3 and other services. Understanding IAM roles/policies is moderate complexity; you should grasp how to create a role with S3 access for Glue. (This is often a stumbling block for new AWS users – expect to spend some time learning how policies work.)
- **Amazon Athena:** A serverless interactive query service that uses SQL to query data stored in S3. Athena is not Spark, but it's complementary – it uses Presto/Trino under the hood. You should know what Athena is good for (ad-hoc queries on S3 data with SQL) versus what Spark is good for (heavy transformations, large-scale joins, etc.). Athena itself is low complexity to learn (it's just SQL on S3 with an AWS console interface), but you should be aware of how it uses the Glue Data Catalog (i.e. why crawling schema is needed) [17] [18].
- **Amazon Redshift:** AWS's cloud data warehouse. In some projects Spark ETL jobs prepare data and load it into Redshift for analytics. Redshift is a managed MPP database; you don't need to become a Redshift expert in 20 days, but know the basics of what it is used for (analytical queries on large data, similar to Snowflake or other warehouses) [19] [20]. It might come up in discussions (e.g. *"When would you use Spark vs. Redshift?"* or *"How do you load data into Redshift?"*). Familiarize yourself with Redshift's purpose and how to connect to it (JDBC or COPY from S3). Complexity is moderate – focus on concepts rather than deep configuration (unless your assessment expects Redshift specifics).
- **Amazon EMR:** AWS's managed Hadoop/Spark cluster service. Although your focus is Glue, be aware of EMR as an alternative for running Spark on AWS [21]. EMR gives more control over cluster configuration. You won't dive deep into EMR in this 20-day plan, but knowing of its existence and basic use (spinning up a Spark cluster on AWS) can show well-rounded AWS knowledge.
- **Terraform (Infrastructure as Code):** Terraform lets you define cloud infrastructure in code. As a data engineer, knowing Terraform is valuable for automating deployment of resources like S3 buckets, Glue jobs, IAM roles, Redshift clusters, etc. [22] [23]. It's a separate skill from Spark but highly complementary in an AWS environment (so you can **provision** the data pipeline components reliably). Expect a learning curve in understanding Terraform syntax and commands (since you're new to it). Start with basic Terraform usage (providers, resources, state). Over 20 days you can attain foundational skill: e.g., create a simple Terraform script to deploy an S3 bucket and an IAM role for Glue [17]. As you progress, you could expand to automating more (Glue crawlers, Athena databases, etc.). Mastering Terraform is an

ongoing journey, but even basic proficiency will impress, given that “Infrastructure as Code” is listed among key skills for AWS data engineers [15].

Skill Hierarchy: Based on the above, here’s a **learning hierarchy** (what to learn first, what builds on prior topics):

- **Foundational prerequisites:** Python basics (focus on data structures, file I/O, lambda functions since Spark uses a lot of functional patterns), SQL (analytical queries, joins – will translate to Spark DataFrame operations), and a refresher on distributed processing concepts (e.g. read a short article or watch a video on “How MapReduce works” and “Why Spark is needed”). These foundations underlie everything else.
- **Core Spark Concepts:** Next, learn Spark’s general architecture and core API. Start with creating a SparkSession and running simple transformations in PySpark on a local or small dataset. Understand RDD vs DataFrame (you can mostly stick to DataFrame API for now). Grasp transformations/actions and lazy evaluation early – it’s a mental shift from imperative programming to Spark’s functional style. Ensure you understand how Spark distributes data via partitions. This stage is critical and of medium complexity – allocate sufficient time.
- **DataFrame API and Spark SQL:** Once basics are down, focus on DataFrames: reading and writing data (CSV, JSON, Parquet) in Spark, performing operations like filter, join, groupBy, aggregations using the DataFrame API, and writing SQL queries through SparkSQL. This is directly relevant to real ETL tasks and leverages your SQL knowledge. Practice on sample data to cement these skills.
- **Spark Application Architecture (Intermediate):** As you start writing Spark jobs, learn about the execution model: e.g. how the driver and executors work, what a task is, and why partitioning and **shuffling** matter. This helps in writing efficient jobs. You don’t need exhaustive detail on the Spark scheduler, but know enough to explain what Spark does when you run a job (builds DAG, splits into stages, tasks on executors, etc.). This knowledge strengthens your foundation and is often something you might be *asked to explain conceptually* in an assessment (to gauge depth of understanding) [24].
- **Spark Advanced Features (optional/awareness):** If time allows, get a high-level view of additional Spark features: **Spark Streaming** (structured streaming usage and how micro-batches work), and performance tuning techniques (caching/persistence, broadcast joins, etc.). These are more complex; treat them as stretch goals. Knowing terminology (like *“streaming”*, *“shuffle”*, *“broadcast variables”*) can be useful in discussion, but you don’t need to master them in 20 days unless you identify a strong need.
- **AWS Integration Basics:** In parallel with Spark learning (once you’re comfortable with the basics), start integrating AWS knowledge. Learn S3 fundamentals (create a bucket, manually upload/download data, understand paths). Then study how Spark interacts with S3 (e.g. Hadoop AWS connectors, setting credentials). Next, learn what AWS Glue does: read about Glue’s architecture (it runs Spark jobs, has development endpoints or interactive sessions, etc.) and how to submit a PySpark script on Glue. Glue also involves configuring connections, using the Data Catalog, and possibly writing code with Glue’s library (which has some convenience wrappers). Also familiarize yourself with setting up IAM roles for Glue – this is a prerequisite to actually run anything on AWS (Glue needs a role with permissions to S3, logs, etc.) [25] [26].
- **AWS Data Warehouse & Query Tools:** Understand at a conceptual level how Redshift fits in (you might read an intro or watch a video on Redshift architecture: leader node, compute nodes, columnar storage, etc. [19]). Also understand how Athena works for querying data on S3. These tools are not Spark, but knowing them solidifies your “data engineer toolbox” and you specifically mentioned you’ll be questioned on them. For example, you should know that Glue can catalog data for Athena, and Athena queries S3 using that metadata; whereas Redshift is a separate system where data needs to be loaded (possibly via Spark or Glue ETL or COPY command).

- **Terraform and Infrastructure as Code:** Once you have a grasp of the AWS services, layer in Terraform knowledge so you can automate them. Start with learning Terraform basics: installing it, initializing a working directory, and writing a simple configuration (provider AWS, resource S3 bucket for example) [23] [27]. Then incrementally introduce more resources: perhaps an IAM role and policy via Terraform (since that's relevant for Glue), and if time permits, define a Glue job or crawler in Terraform. Understanding Terraform's syntax (HCL), state management, and execution plan is the main challenge initially. It's highly recommended to practice by actually running `terraform apply` in an AWS sandbox account to see it work – this gives confidence and practical skill.

The above forms a **skill tree** where early topics (Python, SQL, Spark basics) are prerequisites for later ones (writing complex Spark jobs, using AWS services with Spark, automating with Terraform). Foundational AWS knowledge (S3/IAM) is needed before advanced AWS usage (Glue/Redshift). By mapping these dependencies, you ensure you build knowledge in the right order. In summary, you'll start with Spark fundamentals, then expand to AWS data services integration, and finally cement everything with infrastructure and project-level understanding. This progression will strengthen your data engineering foundation rather than just prepping answers – aligning with the goal of solid competency for your upcoming role assessment.

Step 2: Learning Path Design – Milestones and Sequence

With the core components and prerequisites identified, we can outline a progression tailored to your **beginner level (Spark)** and **experience (BI background)**, all within **20 days (~3–4 hours/day)**. The path is divided into milestones, each with a focus area, ensuring you build knowledge step by step and stay aligned with the 20-day timeframe. The emphasis is on practical understanding, not just theoretical recall, reflecting that the assessment will test your foundational knowledge in a hands-on way rather than in a formal exam setting.

Milestone 1: Big Data Foundations and Environment Setup (Days 1–2) – *Goal:* Get oriented with big data concepts and set up the tools for practice.
- **Topics:** Parallel vs. distributed processing (why we need frameworks like Spark), recall of data engineering fundamentals (ETL, data lakes, etc. from "Fundamentals of Data Engineering"), and environment setup for Spark and AWS.
- **Activities:** Install PySpark locally or set up a simple environment (you can use a Docker image for Spark or just `pip install pyspark` for local mode). If you have access to AWS, configure AWS CLI and an IDE or notebook that can run PySpark. Ensure you have an AWS account ready for later (with appropriate permissions or a budget in mind for Glue/Redshift usage). Refresh Python basics if needed.
- **Time:** ~5–6 hours total. Given your BI experience, you might move quickly through conceptual refreshers. Allocate Day 1 to concepts and Day 2 to environment prep and Python/SQL refresh.

Milestone 2: Spark Core Concepts & PySpark Basics (Days 3–5) – *Goal:* Master the fundamentals of Spark's API and execution model.
- **Topics:** Creating a SparkSession; working with RDDs vs DataFrames; transformations (e.g. `map`, `filter`, `select`, `withColumn`) and actions (`show`, `collect`, `count`); the concept of lazy evaluation and DAG; how Spark parallelizes tasks across partitions. Also cover Spark's data reading/writing basics (loading a CSV or JSON into a DataFrame, and outputting results).
- **Activities:** Follow a PySpark tutorial or course module for beginners. For instance, work through a hands-on exercise like word count or simple aggregations on a sample dataset to experience the transformation/action flow. Experiment in a local PySpark shell or notebook: e.g., read a CSV with `spark.read.csv`, do a simple transformation, and collect results. Visualize or sketch how the operation would be distributed (this helps reinforce understanding of partitions and tasks).

- **Time:** ~9–12 hours. This is a critical section, deserving about 3 days at 3–4 hours each. By end of Day 5, you should feel comfortable writing and running basic PySpark code and explaining how Spark executes it. This corresponds to a major portion of foundational knowledge – it's reasonable to spend ~15% of your total time here as it underpins everything else.

Milestone 3: DataFrame Operations and Spark SQL (Days 6–8) – *Goal:* Become proficient with the high-level DataFrame API for typical data engineering tasks.
- **Topics:** Schema definition and the Spark DataFrame, performing complex transformations (joins between DataFrames, aggregations, groupBy, ordering), using Spark SQL queries, and user-defined functions (UDFs) in PySpark. Also cover handling of different data sources: reading/writing Parquet (columnar format widely used in data lakes), JSON, or others. If relevant, touch on partitioning of output data (e.g., writing out partitioned data by key to multiple files – akin to Hive partitioning).
- **Activities:** Take a sample dataset (perhaps something from Kaggle or a public data source relevant to your domain knowledge) and practice an end-to-end transformation: load raw data, clean or filter it, join with another dataset (if available), and output the result. Write both SQL queries via `spark.sql("...")` and equivalent DataFrame code to get comfortable with both. If following a course, this aligns with intermediate sections of a PySpark course (covering Spark SQL). Ensure you also practice using the Spark DataFrame documentation/API to lookup functions – a vital skill for real projects.
- **Time:** ~9 hours over 3 days. By Day 8, you should reach a milestone where you can use PySpark to accomplish typical ETL data transformations similar to what you might do in a SQL-based ETL tool, but now distributed. This milestone leverages your existing SQL expertise and translates it into Spark, so you might progress faster here, but allow time for debugging and truly understanding the operations.

Milestone 4: Spark Architecture & Advanced Concepts (Days 9–10) – *Goal:* Deepen your understanding of how Spark works internally and learn optimization basics.
- **Topics:** Spark application architecture recap (driver, executors, cluster manager) and how jobs and stages are executed. Learn about **partitioning and shuffling** in more detail – what causes a shuffle (e.g. joins, aggregations) and why it matters for performance. Introduction to performance tuning: using `.persist()`/caching, understanding the impact of data serialization, and maybe the concept of **Spark jobs vs. stages vs. tasks**. Also get familiar with the Spark UI (if you can run Spark locally or use a notebook environment that shows a UI) to monitor jobs – this is very insightful if accessible. Optionally, read about **Spark Structured Streaming** to know how streaming differs (only if interested or if your target projects involve streaming data).
- **Activities:** Read a chapter or article on Spark internals (for instance, on how Spark's DAG scheduler works or what executors do). Try a small experiment: use a large dataset split into many files and see how Spark handles it, or deliberately create a scenario that forces a shuffle (like a join with an unpartitioned key) to observe performance. If possible, enable the Spark UI in local mode to inspect the execution plan (or use `explain()` on DataFrames to see the plan). Write a short explanation for yourself of "What happens when I run a Spark job?" – this will help consolidate your knowledge and prepare you to articulate it during the assessment.
- **Time:** ~6–8 hours. This milestone is shorter but intellectually dense. Allocate 2 days with ~3–4 hours each. By Day 10, you should be able to not only use Spark but also discuss how it works and identify potential performance considerations. This level of understanding will **strengthen your foundation** (fulfilling the goal of deep knowledge over rote answers), and it's often what distinguishes a data engineer who "knows Spark" from one who just followed tutorials.

Milestone 5: AWS Data Engineering Integration (Days 11–14) – *Goal:* Learn how to apply Spark in the AWS ecosystem and understand the key AWS services (S3, Glue, IAM, Athena, Redshift) relevant to data pipelines.
- **Topics:** - **Amazon S3 for Data Lakes:** How to use S3 as a source and sink in Spark. Learn about

configuring Spark to read from S3 (i.e. using `s3a://` URI, AWS credentials through environment or IAM roles). Understand S3 consistency and why, for example, committing output files has some considerations. Also, know how data is typically organized in S3 (raw/staging/processed zones, partitioned folders by date, etc. – concepts you might already know from data engineering fundamentals).
- **AWS Glue Basics:** Understand Glue's components: the difference between a **Glue ETL job** and a **Glue crawler**, and the Glue Data Catalog. Learn how to create a Glue job (through AWS Console or Terraform, etc.) and what configuration is needed (like assigning an IAM role, specifying number of workers or DPUs, etc.). Also, review the minor differences in writing a PySpark script for Glue – for example, Glue provides a `glueContext` and some utilities, but it's largely standard PySpark code with some Glue library extras.
- **Glue Data Catalog and Athena:** Learn to use Glue Data Catalog to store metadata. Perhaps practice by running a Glue crawler on a sample S3 dataset to create a table schema (if you have AWS access). Then see how Athena can query that data. This will solidify how schema-on-read works in AWS.
- **Amazon Redshift:** Read about connecting Spark to Redshift. One common approach is using the **Spark JDBC** driver to write to Redshift (or using the Redshift `COPY` command from S3). Understand at a high level how you would load data into Redshift (for example: Spark saves data as Parquet to S3, then Redshift COPY into table; or using Glue's custom connector). If time and resources allow, you might set up a small Redshift (there is Redshift free trial or you can use Redshift Serverless in a minimal configuration) and attempt a simple load or unload. If that's not feasible, at least conceptually map out the steps to get data from Spark into Redshift.
- **AWS IAM for Glue/S3:** Delve into IAM roles and policies required for data engineering jobs. E.g., what permissions does a Glue job's IAM role need? (Access to S3 buckets, logs, possibly Glue catalog). Practice writing or reading a sample IAM policy. This is important because without the right IAM setup, none of your AWS-based Spark jobs will run. Knowing this also shows you understand cloud security basics.
- **Terraform for AWS Data Services:** Start applying Terraform knowledge here by writing templates for the AWS resources above. For example, author a Terraform script for an S3 bucket and an IAM role for Glue with an S3 access policy [28] [29]. If comfortable, extend it to define a Glue crawler or even a Glue job. The idea is to use Terraform to automate what you've learned to create manually. This reinforces both your AWS knowledge and Terraform skill.
- **Activities:** The focus is on *hands-on practice with AWS*. If you have AWS access and it's within your budget, do the following: create an S3 bucket (via console or Terraform), upload a sample dataset (could be a CSV of something like sales data, which you may have domain knowledge in). Run a Glue Crawler on it (through console or Terraform) to create a Catalog entry. Try writing a simple Glue job in PySpark that reads from the S3 data and does a transformation (even as simple as filtering or aggregating) and writes back to S3 or to another S3 location. Execute that Glue job and verify results. Then use Athena to query the output in S3 (you'll have to create a table in Glue Catalog for the output or use the crawler). Each of these steps solidifies an aspect of AWS data engineering: S3 usage, Glue job authoring, Glue Catalog, and Athena querying. Additionally, practice using Terraform in at least a limited capacity – for example, use Terraform to create the IAM role and S3 bucket before running the Glue job (as was done manually). This can be done in parallel with learning – perhaps write Terraform for resources on one of these days and apply it, verifying in AWS that it created what was expected.
- **Time:** ~12–14 days is allocated across Days 11–14 (4 days, about 12–16 hours). This is a broad milestone covering multiple AWS services, so it needs significant time. It's arguably as important as Spark itself for your goal, since real-world usage will involve these tools. By Day 14, you aim to have completed at least a mini-project using Spark on AWS: for example, *ingest file on S3 -> run Glue (PySpark) job to transform -> land output on S3 (or load to Redshift) -> query via Athena or Redshift*. This hands-on project will not only reinforce learning but also give you something concrete to discuss or demonstrate, showing you can apply Spark in context [30]. Don't worry if not every step is perfect – even setting up and troubleshooting this pipeline is valuable learning (and something you can mention during an assessment to show your problem-solving experience).

Milestone 6: Terraform and DevOps Practices (Days 15–17) – *Goal:* Solidify your ability to automate and manage infrastructure, and understand how it fits into data engineering workflows.
- **Topics:** Dive a bit deeper into Terraform: modules, state, and deploying multiple resources together. Also cover basic DevOps concepts relevant to data engineering: version control (ensure you are using Git to track your code and Terraform scripts), and CI/CD concept (you won't set up a full CI pipeline in 20 days, but know what it means to deploy infrastructure and code through pipelines). Consider how you would deploy a data pipeline in a repeatable way – Terraform for infrastructure, maybe using AWS Glue job scripts stored in S3 or an artifact repository, etc. This milestone connects the dots between just doing things manually and doing them in a production-grade manner. It's not expected you become a DevOps expert, just be conversant in these ideas.
- **Activities:** Expand your Terraform practice: for example, create a Terraform configuration that includes all pieces of a simple pipeline (Bucket, IAM role, Glue job definition, maybe a Redshift instance if daring). Even if you don't apply it (due to cost of Redshift), writing it is a great exercise. Alternatively, simulate by writing Terraform for an EC2 or a dummy resource if you want to avoid costs – the goal is to practice the syntax and process. You can also read case studies or blog posts of AWS data pipelines automated by Terraform to see best practices (there are community examples and Medium articles on this) [31] [32]. Additionally, ensure your project code (PySpark scripts, etc.) is in a Git repo – practice basic Git operations if needed. This is a good point to reflect on folder structure and how you'd organize real projects (scripts, Terraform code, documentation).
- **Time:** ~9 hours across 3 days. By Day 17, you should feel comfortable that you can write Terraform configs for core AWS services used (at least S3, IAM, maybe Glue), and you understand how "infrastructure as code" benefits data engineering (e.g., easier testing, repeatability, avoiding manual errors) [33]. This adds an important dimension to your skill set, fulfilling the request to **focus on Terraform** which is likely to be a talking point in your assessment.

Milestone 7: Review, Project Finalization, and Assessment Readiness (Days 18–20) – *Goal:* Consolidate knowledge, fix any weak spots, and get comfortable explaining what you've learned and done.
- **Topics:** All previously covered topics revisited briefly. Emphasis on areas you feel less confident in. Also prepare for the assessment in a low-stress way: think about how to communicate your understanding of Spark and AWS services clearly. Since the evaluation is not a formal test, focus on being able to discuss fundamentals and perhaps walk through a scenario or your mini-project.
- **Activities:** Day 18 could be used to finish any outstanding parts of your practice project or re-run things that failed initially. Day 19 should be a thorough review day: go back to notes or resources on Spark basics, maybe re-do one of your early exercises without looking at the solution to see if it's second nature now. Create a one-page summary sheet of key Spark concepts (e.g. definitions of RDD, DataFrame, transformation vs action, partition, etc.), and key AWS concepts (what is Glue, how does Spark use S3, etc.). On Day 20, simulate portions of the assessment: for example, try to explain aloud **"How would you design a data pipeline on AWS with Spark for a given use case?"** or **"What happens under the hood when you run a Spark job?"**. You could have a friend or colleague ask you a few basic questions, or even record yourself giving an explanation, to identify any shaky areas. Also, revisit the **why** of each technology: why use Spark instead of just Python scripts (answer: for big data parallelism), why use Glue vs. EMR vs. Redshift for certain tasks, etc. This prepares you to answer high-level questions demonstrating your foundational understanding.
- **Time:** ~9–10 hours over the final 3 days. This includes time for rest and light review to avoid burnout. Importantly, incorporate short **breaks** or lighter sessions in these final days to let your knowledge solidify. The goal is to enter the assessment with confidence and a clear mind, rather than trying to cram new information. You've built a strong foundation by now, so this stage is about polish and confidence.

Each milestone naturally builds on the previous ones, following an optimal learning sequence from basic concepts to integrated skills. The timeline above respects your **20-day, ~70 hour** constraint by prioritizing critical topics (Spark and AWS fundamentals) and trimming optional ones (Spark streaming, deep dive into Redshift, etc.). It ensures by the end of the period, you not only **know** the material but have also applied it in practice, aligning perfectly with the goal of strengthening foundations for your new data engineering role.

Step 3: Resource Curation – Books, Courses, and Tools (Mixed Learning Style)

To maximize learning in a short time, we'll use a **mix of resources**: video courses for interactive learning, written material for depth and reference, and hands-on tutorials/projects for practical experience. Below is a curated list of high-quality resources, organized by category and priority, tailored to cover Apache Spark, AWS integration, and Terraform. Each resource has been chosen for effectiveness and alignment to your goals, so you can pick what suits your learning style or even use them in combination:

- **Book / Official Guide:** *Learning Spark, 2nd Edition* (O'Reilly, by Jules Damji et al.) – An excellent up-to-date book covering Spark 3.x. It explains core concepts (RDD, DataFrame, Spark SQL, etc.) with examples. Databricks provides a free PDF download [34], making it easily accessible. This book is great for deepening your understanding and as a reference. **Priority:** High – read relevant chapters alongside your hands-on practice (e.g., read the DataFrame chapter during Milestone 3). It's well-regarded in the community and even recommended for Spark certification prep [34], so it aligns well with foundational learning.

- **Video Courses (PySpark & Spark fundamentals):**

- *"Spark and Python for Big Data with PySpark" by Jose Portilla* – A popular Udemy course covering PySpark from basics to advanced. It includes hands-on labs (like word count, movie ratings analysis) and is well-suited for beginners. This course will give you practical coding experience in Spark and is quite engaging [35]. **Priority:** High – Use this as a primary guide in Milestone 2 and 3; it will walk you through setting up PySpark and applying transformations.
- *"Taming Big Data with Apache Spark and Python" by Frank Kane* – Another Udemy course with a hands-on approach. Frank Kane explains concepts clearly and has you build real use-case projects (e.g., analyzing social network data). It's slightly older but still very relevant, especially for core Spark concepts [36]. **Priority:** High if available – You might choose between this and Portilla's course, or skim both. Frank Kane's content on RDDs and optimizations is valuable for Milestone 2 and 4 (he discusses how Spark works under the hood in an easy-to-understand way).
- *"Apache Spark 3 – Spark Programming in Python for Beginners" by Prashant Kumar Pandey* – A course focused on Spark 3.x and PySpark [35]. This could be an alternative or supplement if you want more explanation on certain topics. **Priority:** Medium – use it if you find you want additional perspective or practice problems beyond the above two.

- *(Optional) "Data Engineering Essentials Hands-on – SQL, Python, and Spark" by Durga Gadiraju* – This course covers a broader range (including SQL and Python) along with Spark [37]. Given you already have a BI background, you might not need the SQL/Python parts, but the Spark portions and exercises could be useful for reinforcement. **Priority:** Low/Optional – consider for extra practice if time permits or if you prefer the instructor's style.

- **AWS and Glue Learning Resources:**

- *AWS Official Documentation and Workshops:* The AWS **Glue Developer Guide** and **Glue Workshop** tutorials on AWS's website are excellent for learning specifics of AWS Glue. For example, the AWS tutorial *"Writing an AWS Glue ETL script"* walks you through creating a Glue job, and the **Getting Started with AWS Glue** (DataCamp or AWS Workshop Studio) shows the process of crawling data and writing a job [38]. These official docs ensure you get the details right (like IAM roles, script structure). **Priority:** High – Use official docs as reference when you actually set up Glue jobs or encounter issues. Skim through the Glue Developer Guide sections on "Jobs," "Crawlers," and "Connections."
- *Video:* "PySpark for AWS Glue Tutorial [Full Course in 100 min]" on YouTube [39] – A compact video that demonstrates using PySpark with AWS Glue end-to-end. It likely covers setting up a Glue job, writing a PySpark script, and running it. This can be very helpful to watch before you attempt your own Glue job, as it highlights common gotchas and AWS console steps. **Priority:** Medium – 100 minutes is a good investment to see the whole picture; consider watching this around Day 11-12 as you enter the AWS integration phase.
- *Pluralsight Course: "AWS Glue":* If you have Pluralsight access, there's a dedicated course that covers Glue, including Glue Data Catalog, PySpark in Glue, and deployment of Glue jobs [40]. It's structured and can supplement your hands-on practice. **Priority:** Medium – not essential if you're hands-on, but beneficial if you prefer a guided walkthrough of Glue features.

- *AWS Whitepapers/Blogs:* AWS often has blog posts or architecture blogs about building data lakes, using Glue with Redshift, etc. For instance, *"Build an ETL pipeline to load data from S3 to Redshift using Glue"* (AWS Big Data Blog) is a step-by-step that might mirror what you want to do [41]. Also, consider reading *"AWS Data Engineering Roadmap"* [42] [14] for a high-level overview of skills – it reinforces why we chose to focus on certain services.

- **Apache Spark Reference & Community:**

- *Official Apache Spark Documentation:* Especially the sections on **Spark Programming Guide** (for RDDs/DataFrames) and **Configuring Spark**. The official docs can be dense but are very authoritative. Use them to clarify specifics (like what certain transformations do, or how to configure S3 access).
- *Databricks Free Community Edition:* Though you are focusing on AWS, practicing Spark on Databricks CE is a convenient option (browser-based notebooks with a free small Spark cluster). It could be an easy way to try out Spark code without local setup issues. **Priority:** Low/Optional – if your local PySpark or AWS environment has issues, this is a good fallback for Spark practice (just remember the deployment on AWS Glue will be slightly different).

- *Stack Overflow and Spark Summit Talks:* If you get stuck on a Spark error or need to optimize something, sites like Stack Overflow are invaluable – search error messages or questions (likely someone has asked before). Additionally, certain YouTube talks from Spark Summit (or Data+AI Summit) can deepen understanding (e.g., talks on Spark best practices, common pitfalls). Use these as needed when you encounter challenges or for deeper insight after covering the basics.

- **Terraform & DevOps Resources:**

- *Official Terraform Tutorial (HashiCorp Learn):* HashiCorp's official tutorials (on learn.hashicorp.com) for AWS are very beginner-friendly. Start with "Terraform AWS Provider – Getting Started" and "Building AWS EC2 and S3 with Terraform" to grasp basics. **Priority:** High – complete at least the introductory tutorial to get familiar with Terraform commands and syntax.
- *"Terraform for Data Engineers" article series* by Data Dev Backyard [43] [31] – A Medium series specifically targeting Terraform use in data engineering (covering S3, IAM, Glue, Athena via

Terraform). The series includes step-by-step examples (with code) to create a data lake infrastructure as code. For example, one part shows creating an S3 bucket and uploading data, another shows setting up a Glue crawler and Athena database via Terraform [17] [44]. **Priority:** High – these are directly relevant to your use case. You can follow along these tutorials in the later part of your learning path (Milestone 5 and 6) to solidify Terraform skills and simultaneously build your project.
- *YouTube:* "Automate S3 Data ETL Pipeline with AWS Glue and Terraform" [45] – A video tutorial that pairs well with the above Medium series. It walks through using Terraform to deploy an AWS Glue-based pipeline. Watching an expert go through the process can clarify the interplay between AWS services and Terraform. **Priority:** Medium – if you prefer a visual/demo learning style, use this in Milestone 6 to reinforce what you read in the articles.
- *Terraform Documentation:* Reference the Terraform AWS provider docs when writing your configs (e.g., docs for `aws_glue_job` resource, `aws_iam_role`, etc.). These will be the source of truth for which arguments to use. They're dry reading, so use them as lookup references rather than cover-to-cover.

- *CI/CD for Data Engineering:* Since you mentioned interest in Terraform and being questioned on it, they might also touch on general DevOps practices. A worthwhile read is a blog or short course on using Git and CI/CD for infrastructure. For instance, *"Mastering AWS Data Analytics with Terraform and GitLab"* [46] might provide insight into how code and infrastructure are deployed together. **Priority:** Low – good to skim if curious, but not critical to finish within 20 days; focus more on getting Terraform basics right.

- **Practice Projects and Exercises:**

- *Data Engineer Things – AWS Glue Pipeline Project* [47] – The blog by Bella Jiang on building an ETL pipeline with Glue and Terraform (which we referenced) is essentially a ready-made project scenario. It details an end-to-end solution (extract from S3, transform with Glue, load to Redshift, plus Athena for querying). Use this as a blueprint for your own project. You can follow the same structure with a simpler dataset. **Priority:** High – use this as a guide for Milestone 5 practice. Even if you don't replicate it exactly, reading through the steps and code will teach you a lot about real-world implementation.
- *Kaggle or Google Datasets:* Find a moderately sized dataset (a few tens of MB to a few GB) and use it for Spark exercises. E.g., Kaggle's "Seattle Rainfall" or "MovieLens ratings" – something you find interesting. This makes practicing more engaging. **Priority:** Medium – not a resource to read/watch, but a source of data for hands-on practice which is crucial for mastery.
- *Community Forums:* Consider joining the r/dataengineering subreddit or relevant Stack Exchange communities. Seeing what questions others ask (and their solutions) can be enlightening. For instance, the Reddit thread we looked at had great pointers on what to focus on (SparkContext, partitions, etc.) [13] [24]. **Priority:** Low – use as supplementary motivation or problem-solving help if needed.

Resource Utilization Plan: To avoid overwhelm, pick a primary learning resource for each topic and use others to supplement: for example, use Jose Portilla's course as your main Spark intro, supplement with *Learning Spark* book for theory and deeper dives, and consult the official docs or Stack Overflow when you hit specific questions. For AWS, perhaps use the Glue official tutorial as a guide while also watching the YouTube Glue tutorial for clarity. For Terraform, follow the Medium series hands-on, and refer to HashiCorp docs for any syntax doubts.

Remember that **quality trumps quantity** – it's better to fully engage with a few resources than skim many. The list above is comprehensive for reference, but you don't have to exhaust all of it in 20 days.

Focus on those marked high priority first. Also, since your learning style is mixed, don't hesitate to switch formats: if you feel saturated reading text, watch a video to reinforce the concept (and vice versa). The combination of seeing, reading, and doing will reinforce learning via multiple channels, leading to better mastery.

Lastly, maintain a notes document or journal as you go through resources. Write down key points (especially AWS service limits, Spark trickiness, Terraform commands) and things to remember. This will be invaluable during quick reviews and for formulating answers during your assessment. Every resource you use, try to directly connect it to something you might need to do on the job or answer in an interview – this keeps the learning purposeful and focused.

Step 4: Practice Framework – Exercises, Real-World Scenarios, and Reinforcement

To **truly master** these skills and solidify your foundation, practical application is key. Below is a structured practice plan with exercises for each topic area, real-world scenarios to simulate on your own, and a **spaced repetition** approach to reinforce learning over the 20 days. The idea is to learn by doing and to revisit concepts at intervals so they stick.

Topic-wise Exercises:

- **Python & SQL Refresher (Day 1–2):** If needed, write a short Python script to parse a simple file (e.g., a CSV of sales records) and compute a summary (total sales per category). This warms you up for thinking in data terms. For SQL, take an example dataset (maybe your company's sample data or an open dataset) and write a few non-trivial queries (joins, group bys). *Checkpoint:* You can correctly write a query to answer a business question from data. This ensures your base skills are sharp before diving into Spark.

- **Spark Basics (Day 3–5):** After learning a concept in Spark, implement it immediately:

- Create an RDD from a small list and practice transformations: e.g., map a list of numbers to their squares, filter out even numbers, reduce to a sum. Then do the same using the DataFrame API (if possible, e.g., create DataFrame from a list of values) to see the difference.
- Use a real file: for instance, take a CSV (perhaps `airline_delays.csv` or similar public data). Use **DataFrame API** to read it (`spark.read.csv` with header and inferSchema). Practice basic ops: show schema, print a few rows, filter for a condition, select certain columns, etc.
- After each operation, consciously think: *Did this run a transformation or an action?* If transformation, check Spark UI or use `df.explain()` to see that it's building a plan (optional but insightful). If action, note that execution happened.

- Write a simple PySpark script (not in interactive mode) that reads an input file and writes output to disk (or memory). Run it using `spark-submit` in local mode if you can, to simulate running a job end-to-end. This script could, for example, read a CSV of log entries, filter by a criterion, and output another CSV of results. *Checkpoint:* By end of these exercises, you should be able to write a basic PySpark job without looking up every step, and you understand how to run it (even if just locally). You should also be able to explain what an RDD is and what a DataFrame is in simple terms.

- **DataFrame & Spark SQL (Day 6–8):** Practice with a **realistic dataset** to perform multi-step transformations:

- Use two related datasets (e.g., "Orders" and "Customers" as CSVs). Load each as a DataFrame. Perform a join (e.g., attach customer info to orders). Then do an aggregation on the joined data (total orders per region or per customer segment). Do this using DataFrame API (with `.join()`, `.groupBy()`, etc.) and verify results by also writing a SQL query using `spark.sql` (you can create temporary views for the DataFrames and query them). Ensure both approaches give the same outcome.
- Try writing a **UDF** (User Defined Function) in PySpark: e.g., a small Python function to categorize a value ("high", "medium", "low") and use it on a column via Spark. This will teach you about serialization and that UDFs can be slower – note the performance but it's fine on small data.
- Work with **varied data formats**: If you've been using CSV, try reading a JSON or Parquet. For Parquet, note how schema is preserved and it's faster – a common format in data lakes. If you have time, also try writing output in Parquet format and then reading it back (good practice for understanding how Spark writes partitioned files).
- *Real-world scenario:* Imagine a scenario: "Combine website clickstream data with user profile data to produce a daily active users report per region." Simulate this: one DataFrame = clickstream (userId, timestamp, etc.), another = user profiles (userId, region). Perform a join and group by region. Even if the data is small, treat it like it's big and you needed Spark. This will help frame your thinking for business problems.

- *Checkpoint:* You can perform joins and aggregations in Spark and get correct results. You should also be comfortable using Spark's API docs or quick Google searches to find how to do a new operation (this is normal in real work). At this point, try to do one task **without peeking** at notes or examples – e.g., "In the orders dataset, find top 5 products by sales". If you can code this in PySpark end-to-end solo, you're on track.

- **Spark Architecture & Optimization (Day 9–10):** Here the "exercises" are more conceptual, but make them hands-on where possible:

- Take a moderately large dataset (something with, say, 100k–1M records; if not available, you can fake one by duplicating data until it's sizable). Time a simple operation (like count or filtering) and note it. Now repartition the DataFrame to a higher number and time it again (maybe on multiple cores if you have them). This can illustrate how partition count affects performance.
- Induce a shuffle intentionally: e.g., group by a high-cardinality column. Look at the `explain()` output – identify the "exchange" or "shuffle" in the plan. This connects theory to actual jobs.
- If possible, run the Spark UI (if using Databricks CE or another environment that shows the UI for your job) to inspect one of your jobs. If not, use `spark.sparkContext.statusTracker()` or logs to see some info about tasks.
- Write a small paragraph (for yourself) describing the flow of a Spark application: from SparkSession creation -> job -> stages -> tasks on executors -> result to driver. Use the correct terminology. This exercise of articulating it will test your understanding. If there are gaps, revisit resources or ask a question on a forum to clarify.

- *Checkpoint:* You should be able to answer questions like "What is a partition in Spark?" "Why is avoiding shuffles important?" "How does Spark achieve fault tolerance?" (e.g., via RDD lineage and recomputation) [48] [49]. You don't need word-for-word perfection, just ensure you conceptually understand these. Maybe use flashcards for terms (e.g., one card "DAG" – you recall "Directed Acyclic Graph, Spark's way of representing the job for optimization and fault tolerance"). Spaced repetition of these flashcards every few days (Day 10, 14, 18) will keep them fresh.

- **AWS Integration Exercises (Day 11–14):** This is the core of applying Spark in context:

- **S3 and Glue Hands-on:** Start by using boto3 or AWS CLI to put a data file on S3 (just to get familiar if you haven't). Then on AWS Glue, create a simple script. You can actually use the same Spark script you ran locally earlier (e.g., the join or aggregation task) and adapt it to run on Glue. This might involve minor changes like using `GlueContext` if needed and ensuring the script can get input from S3 and write to S3. Use a small subset of data to test. Running the Glue job will be a trial-and-error process (permissions, script errors, etc., are common) – treat each error as a learning step. Document for yourself what went wrong and how you fixed it (this is practice for problem-solving which is key to data engineering).
- **Glue Crawler and Athena:** If you have data on S3 (say the output of your Glue job), set up a Glue crawler to create a table. Verify in AWS Glue Catalog that the table schema is created. Then go to Athena and try a SELECT * from that table. If it returns results, congratulations, you have created a mini data lake! This exercise ties together multiple AWS services (S3, Glue, Athena) and is exactly the kind of real-world scenario data engineers deal with.
- **Redshift Loading (Optional if resources allow):** Spin up a small Redshift cluster (or use Redshift Serverless free tier if available). Use the Glue job (or a Python script) to COPY data into Redshift. Alternatively, use Spark's JDBC to write to Redshift (there are JDBC drivers for Redshift that Spark can use). Even if you do this on a tiny scale, it's good practice to see how Spark might interact with a warehouse. If you cannot use an actual Redshift, simulate by using your local PostgreSQL: write a Spark DataFrame to Postgres via JDBC. This is similar in concept and will teach you about driver jars and connection strings.
- **Terraform Automation:** For each manual step above, consider how to automate it. For practice, after you've done something via the AWS Console, attempt the Terraform equivalent: e.g., after manually creating an S3 bucket and IAM role for Glue, write the Terraform config for those and see if it can create another bucket or so. If you successfully ran a Glue job via console, try defining that Glue job as a Terraform resource (you can use a dummy script or the same script, and see if Terraform can deploy it). This exercise helps you learn Terraform by directly applying it to what you just learned. It's also a form of repetition: you're revisiting the concepts (S3, Glue) in a different way, which reinforces memory.
- *Real-World Scenario:* Define a mini-project scenario to guide these exercises, for example: *"Build a data pipeline that takes daily sales data from an S3 landing zone, cleans it in Spark, and loads a summary to a Redshift data warehouse, with all infrastructure as code."* While you might not fully implement every part in 20 days, framing your practice with this story in mind makes it cohesive. You essentially did this in pieces: ingest (put data to S3), process (Glue/Spark job), store (Redshift or another DB), catalog/query (Crawler & Athena), and infra (Terraform).

- *Checkpoint:* By the end of these AWS exercises, you should be able to answer: "How do I run a Spark job on AWS?" and "How do I set up a data pipeline on AWS end-to-end?" in a step-by-step manner [30]. You will also have encountered common issues (e.g., IAM permission denied, data format problems, etc.) – which is good! It's better to face them now in practice so you're not thrown off by them later. To reinforce learning, write a brief summary or create a diagram of the pipeline you built (show S3 -> Glue job -> S3/Redshift -> Athena). Revisit this summary a few days later as part of spaced review.

- **Terraform & DevOps Practice (Day 15–17):** Continue building on Terraform:

- Take the Terraform scripts you wrote in earlier exercises and refactor them into modules or a cleaner structure. For instance, if you wrote inline IAM policies, see if you can use Terraform modules or at least separate out variables. This will deepen your understanding of Terraform syntax and best practices.

- Perform a **Terraform destroy** and re-apply cycle (in a safe environment) to see that your infrastructure can be torn down and recreated reliably. This is a confidence booster that you truly have IaC under control.
- If you use Git, try pushing your Terraform code to a private repo. Maybe simulate a CI run by simply running `terraform plan` again in a fresh clone of the repo – this mimics deploying infra from code.
- Think of potential interview/practical questions like "How would you use Terraform in deploying a data pipeline?" and ensure you have a concrete answer (you do, from experience now). Possibly practice writing a Terraform snippet on paper or whiteboard (for an IAM role, etc.) to test your recall of syntax – you don't need to memorize everything, but knowing the general structure (provider, resource blocks, etc.) without always checking docs is a plus.
- *Checkpoint:* You can comfortably say you've *used* Terraform to create AWS resources. You know the basic commands (init, plan, apply, destroy). You can discuss the benefits of IaC for data engineering (reproducibility, easier collaboration, avoiding manual errors) [33] . For spaced repetition, revisit your Terraform code a few days after writing – do you still understand it? Try explaining what each part does as if teaching someone; teaching is a great test of mastery.

Spaced Repetition & Review Strategy:
The plan naturally revisits key themes (Spark basics, AWS basics) multiple times. To formalize this: - Every **3–4 days**, take 30 minutes to review notes or flashcards from earlier topics. For example, on Day 4 review Day 1–2 concepts (what is MapReduce, etc.), on Day 8 review Spark basics from Day 3–5, on Day 12 review some Spark SQL or architecture points from earlier, and so on. This ensures nothing is forgotten as you move forward.
- End of each week (Day 7, Day 14) do a **mini-quiz or recap** for yourself. This can be a simple list of questions you try to answer without help. E.g., *"List the steps to run a PySpark job on AWS Glue."* or *"What does* `.persist()` *do in Spark and when would you use it?"*. If you find you struggle to answer, flag that topic for extra attention.
- Leverage multiple learning modes in repetition: re-watch a short segment of a course at higher speed as review, or re-read a summary article. Sometimes hearing/reading it again after doing it helps cement knowledge (you'll pick up nuances you missed the first time).
- Keep using what you learned in later exercises. The practice plan is cumulative – e.g., when doing AWS tasks, you're implicitly revisiting Spark and Python knowledge. Continue to write Spark code during AWS phase so those skills stay sharp. Conversely, while focusing on Spark, periodically think "how would I deploy this on AWS?" – even if you haven't done it yet, this primes you to connect the dots when you reach the AWS part.

Progress Checkpoints and Self-Assessment:
Built into the exercises are "checkpoints" which serve as progress markers. Treat them seriously: when you hit one, if you don't feel confident, pause and reinforce before moving on. Also use these checkpoints to gauge if you're ahead or behind schedule and adjust. For example, if by Day 5 you're still struggling with basic Spark syntax, consider spending an extra day on it (perhaps borrowing from the advanced Spark time) because those fundamentals are non-negotiable. On the other hand, if you sail through Spark basics, you can advance sooner to AWS integration where you might want more buffer time. The plan should be dynamic to your pace – just keep the end goals in mind.

By following this practice framework, you'll engage in active learning and continuous reinforcement. Each exercise bridges theory to a real-world-like task, which not only prepares you for an assessment but for actual job duties. The spaced repetition ensures that by Day 20, you haven't forgotten Day 1's lessons – instead, you have built a durable understanding. As you practice, always think: *"How would I explain this in an interview or to a colleague?"* – this mindset will make your learning more targeted and effective for demonstrating your knowledge when the time comes.

Step 5: Progress Tracking System – Metrics and Feedback Loops

To ensure you're on track and to measure your growth over the 20-day plan, we'll establish a progress tracking system. This includes **measurable indicators**, **self-assessment criteria**, and **feedback loops** to adjust your learning as needed. The goal is to make your progress visible and to keep you motivated by recognizing incremental achievements.

Measurable Progress Indicators: We will use both quantitative and qualitative metrics: - **Completion of Milestones & Exercises:** Track whether you have completed the key exercises/practice tasks for each milestone. For example, *"Completed Spark join and aggregation exercise (Y/N)"* or *"Deployed a test Glue job on AWS (Y/N)"*. These binary completions are straightforward indicators. You can list all planned tasks and check them off as done.
- **Time Spent vs. Planned:** Keep a log of hours studied/practiced each day. Given ~3–4 hours per day is the target, note if you hit that. If you consistently go under or over, adjust the schedule or your pace. For instance, if by mid-week you only hit 50% of planned hours, you may need to allocate an hour extra on the weekend or trim some optional content.
- **Knowledge Quizzes/Flashcards Performance:** If you use flashcards or end-of-week quizzes (as suggested in practice framework), track your score or recall rate. E.g., out of 10 concept questions on Spark architecture, you could answer 8 confidently. Improvement in this score over time is a measurable sign of learning.
- **Project Progress Metrics:** Define small milestones in the project (like *"Data successfully loaded to S3"*, *"Glue job ran without errors"*, *"Terraform script applied without issues"*). Mark dates when each is achieved. If something is not achieved by the expected date, that's a signal to troubleshoot or seek help.
- **Confidence Level Ratings:** Though subjective, rate your confidence in each major topic at the start, mid-point, and end (scale of 1–5). For example: Day 1 Spark knowledge = 1 (novice), Day 10 Spark knowledge = 3 (comfortable with basics), Day 20 Spark knowledge = 4 (ready to apply/discuss). Seeing these self-ratings improve can be encouraging and highlight areas needing extra work if a rating lags.

Assessment Criteria (What "Success" Looks Like): Define what it means to have a solid foundation for each component: - **Spark Fundamentals:** *Criteria:* You can articulate key Spark concepts (like Spark vs. Hadoop, role of driver/executor, meaning of partition) in simple terms, and write a basic PySpark job without external help. *Measurement:* Successfully explained Spark in a mock conversation (e.g., to a colleague) and wrote a script that runs correctly.
- **Spark Advanced/Optimization:** *Criteria:* You can identify at least 2-3 common performance considerations (e.g., "avoid shuffles by partitioning or using broadcast joins when appropriate") and have experienced using Spark UI or explain plans. *Measurement:* Wrote down optimization techniques from memory and verified them against sources; optimized a test job (like reduced run time or number of shuffles) intentionally.
- **AWS Glue & S3 Integration:** *Criteria:* You can set up and run a Glue job that reads/writes S3, and you understand Glue-specific concepts (catalog, crawler, role). *Measurement:* Demonstrated by actually running a job and getting expected output on S3; can list the steps to configure a new Glue job (from role creation to job execution) without looking it up.
- **AWS Athena/Redshift usage:** *Criteria:* You know how to query data on S3 via Athena and how data would be loaded into Redshift, with an understanding of when to use each tool. *Measurement:* Executed a query in Athena on your dataset; wrote a short plan for loading data into Redshift (or tested with Postgres) and can explain it.
- **Terraform & IaC:** *Criteria:* You can write Terraform config for basic resources and explain the deployment process. *Measurement:* Successfully used Terraform to create at least one AWS resource (seen it work in AWS account); able to show a snippet of your Terraform code as an example of your work.
- **Soft Skills and Communication:** (Often overlooked in tracking, but important for assessment) *Criteria:*

You can confidently discuss your approach, explain technical concepts understandably, and maybe even relate it to business value (given your BI background, you can connect the tech to the business). *Measurement:* Did a mock Q&A or presentation of your project to a friend and received feedback that it was clear. If no peer available, record yourself and see if the explanation is coherent and within a reasonable time (say 5-minute explanation of your project).

Feedback Loops: Since you might be studying mostly solo, set up ways to get feedback: - **Self-Reflection Journals:** Every 2–3 days, write a short entry: what did you learn, what was challenging, what do you plan to do to address it? The act of writing this down is feedback to yourself and can reveal patterns (e.g., "Still having trouble with Spark DataFrames API – maybe find another tutorial or ask for help").
- **Peer or Mentor Check-ins:** If possible, arrange a check-in with a colleague or friend in the data field around Day 10 and Day 20. Explain to them what you've learned so far and ask if it makes sense. Encourage them to ask questions. Their questions will highlight if there are areas you're shaky on. Since the expected evaluation isn't strict, even an informal chat with someone can simulate it. If you don't have a person available, even an online forum (Reddit r/dataengineering, Stack Exchange) can serve: try answering a newbie Spark question on a forum – if you can answer others' questions correctly, that's great feedback that you know the material.
- **Mock Assessment near Day 20:** Design a mini-assessment for yourself on Day 19 or so. For example, set a task "Using Spark, read this data and give me X insight, and describe how you'd productionize it on AWS." Then actually do it: write the code, and verbally describe the solution as if presenting. Compare your solution to best practices (you might use your resources or notes to see if you missed something). This mock run will highlight any last-minute gaps.
- **Milestone Retrospective:** At the end of each milestone (as outlined in Step 2), do a quick retrospective. Ask: *Did I achieve this milestone's goal? What was the most difficult part of this section? Do I need to revisit anything later?* Keep a "revisit list" – topics you want to come back to in review. Incorporate that into your spaced repetition schedule. For example, if after Milestone 3 you note "Still not clear on Catalyst optimizer," schedule a review on Day 15 for that specifically.

Progress Tracking Template: Create a simple table or spreadsheet to track the above. For instance:

Date	Planned Task/Topic	Done?	Notes/Issues encountered	Confidence (1-5)
Day 3	Learn Spark basics, run wordcount		Had trouble with SparkContext at first, resolved after reading docs.	3 (improving)
Day 5	DataFrame join & aggregation exercise		Join worked, but aggregation gave nulls – fixed by handling missing values.	4
Day 11	Setup AWS Glue job on sample data		Needed to adjust IAM role; job succeeded with small dataset.	3 (Glue is new, needs practice)
Day 14	Glue -> S3 -> Athena pipeline end-to-end	⚠ Partially	Crawler created table, Athena query works. Haven't loaded to Redshift yet.	4 (Athena good, Redshift pending)
Day 18	Full review and project presentation	–	(Planned)	–

In the above, the “Done?” column is a checkmark or status, and notes capture any learning issues (which you can address). The confidence column is your self-assessment. This template helps you visually see progress. For example, by mid-point, a lot of means you’re on track; multiple ⚠ or (not done) means you might need to adjust the plan or spend extra time.

Milestone Completion Metrics: We define that a milestone is complete not just when tasks are done, but when understanding is achieved. For each milestone, you could have a one-line metric like: *“Milestone 2 (Spark Core) complete: Can explain lazy evaluation and executed 3 transformations and 2 actions in code (Yes/No).”* Ensure the answer is “Yes” before truly moving on. If “No,” plan a remedial action (like revisit a chapter or do one more exercise). This metric tied to understanding ensures you don’t rush ahead with incomplete knowledge.

Adapting to Feedback: The system isn’t just to record progress, but to inform changes. If your tracking shows you struggling with a topic, you can adjust your study schedule (Step 6) accordingly – maybe allocate an extra day, or find another resource for that topic. For instance, if by Day 8 your confidence in Spark DataFrames is still 2/5, you might pause new material and do an extra day of practice or watch another tutorial on that specifically. It’s better to solidify as you go than to have a big gap at the end.

By diligently using this tracking system, you will create a feedback loop: **Learn → Apply/Assess → Identify Gaps → Adjust Learning**. This iterative approach will greatly improve your mastery of each topic. Moreover, having a journal of your progress and achievements can boost your confidence – you can look back and see how far you’ve come in just 20 days. It will also serve as notes you can quickly skim right before your assessment to remind yourself of key points and successes (“Yes, I solved that problem, I can definitely handle the questions around it”).

Step 6: Study Schedule Generation – 20-Day Action Plan

Finally, we break down the entire learning path into a **daily/weekly study schedule** that fits your 20-day timeline and 3–4 hour daily commitment. This schedule balances theory and practice, includes rest/review periods, and aligns with the milestones from Step 2. Below is a detailed plan outlining what to focus on each day. (Feel free to adjust exact days as needed; the key is the sequence and allocation of time.)

Week 1: Spark Fundamentals Immersion (Days 1–7)

- **Day 1: Big Picture & Setup** – *Duration: ~3 hours.*
- Morning (1h): Read a high-level overview of Apache Spark (what it is, why it’s used) – e.g., the introduction of *Learning Spark* or a blog post. Also refresh big data concepts: watch a short video on MapReduce vs. Spark for context [13] [24].
- Afternoon (2h): Set up your environment. If using local machine, install PySpark (verify by running `pyspark` shell). If possible, also set up an AWS free tier account (you won’t use it heavily yet, but get access keys configured, perhaps install AWS CLI and run a test `aws s3 ls`). This is also a good time to organize your learning workspace: create folders for code, a place for notes.

- *Checkpoint:* Environment ready (can run a simple `print("Hello Spark")` in a PySpark shell). You have a clear idea of why Spark is important in data engineering.

- **Day 2: Python & SQL Refresher; Dive into Spark Basics** – *Duration: ~4 hours.*

- Morning (1.5h): Briefly practice Python – write a sample script as mentioned in Step 4 (e.g., parse a small CSV, do a simple aggregation). Similarly, run a couple of SQL queries on a sample dataset (you can use your Postgres Docker or an online SQL sandbox) to ensure your SQL muscles are active.
- Afternoon (2.5h): Start a Spark course (Jose Portilla or Frank Kane). Cover the introductory modules: typically an overview of Spark and setting up the first RDD/DataFrame. Follow along with coding examples. Run the first exercise (like a word count or simple transformation example) on your machine. Get comfortable with using Spark shell or notebook.
- *Review (0.5h in the evening):* Jot down key terms from today (SparkContext, RDD, etc.) and ensure you understand them. This is a light review to reinforce day 2 content.

- *Checkpoint:* You've executed your first PySpark code and understood it. Python/SQL feels comfortable, not a blocker.

- **Day 3: Core Spark API – Transformations & Actions** – *Duration: ~3–4 hours.*

- Focus: Learn about transformations (map, filter, etc.) and actions (collect, take, save). Use your course or *Learning Spark* chapters on RDD/DataFrame basics.
- Practice: In a PySpark shell or notebook, create a simple RDD from a list and apply a few transformations and an action, as per the practice plan. Then create a DataFrame from a Python list or small pandas DataFrame, do a similar operation. Compare the two approaches (just for conceptual clarity).
- By end of session, try a slightly bigger example: maybe use a public small CSV file (load it with `spark.read.csv`) and do a couple of operations and `show()` the result.

- *Checkpoint:* You can explain what a lazy transformation is versus an action in Spark [9] [11], and you've seen that Spark doesn't execute until an action is called.

- **Day 4: DataFrame API Deep Dive** – *Duration: ~3 hours.*

- Focus: DataFrames and Spark SQL. Use a tutorial or book chapter focusing on DataFrames. Learn how to select, filter, join, aggregate using the DataFrame API.
- Practice: Load a dataset (e.g., use two small CSVs as planned). Write code to join and aggregate. Try writing the same logic in SQL (register temp views and use `spark.sql`). This will likely take some trial and error – which is good. By struggling through a join or groupBy, you'll learn how to use documentation or fix errors (like ensuring you handle data types or missing values).
- If time permits, experiment with a Spark SQL function you haven't used before (say, a date function or string function) to see how to apply it.
- *Evening (0.5h):* Quick review of Days 3-4 concepts (maybe flashcards or explaining to yourself what you did).

- *Checkpoint:* By now, the idea of a DataFrame in Spark should feel akin to a table in SQL – you can manipulate it with various operations. You should also be comfortable with the idea that the DataFrame API and Spark SQL achieve the same result under the hood.

- **Day 5: Spark Advanced API & Tuning Basics** – *Duration: ~3–4 hours.*

- Focus: Cover remaining core Spark topics such as repartitioning/coalescing, caching, and possibly an intro to Spark's architecture (driver/executor concept). Your course might have a section on Spark architecture or performance – watch/learn that.

- Practice: Take the DataFrame from Day 4 and play with `df.repartition()` to see effect. Try caching it (`df.cache()`) before an action and see if reusing it is faster (on a small data set it might not be noticeable, but the practice of caching is learned).
- Also, today, run a Spark job in **"local[4]"** mode (if you have 4 cores) to simulate parallelism. See if you can notice CPU usage across cores.
- Read about a common Spark issue like the small files problem or data skew (even if just conceptually). This prepares you for understanding why partitioning strategies matter.

- *Checkpoint:* End of Week 1 core Spark learning – Can you summarize how Spark executes a job? Try writing it out or telling someone (cover: SparkContext, job -> tasks on partitions on executors) [50] [51]. If you can, you're in a good spot. If not, note the gaps and plan to review them on Day 7.

- **Day 6: Mini-Project and Recap** – *Duration: ~3 hours.*

- It's time to consolidate what you've learned in a slightly more open-ended way. Choose a mini-project (maybe the clickstream + user data scenario, or any data you find interesting). Design a pipeline with 2–3 steps of transformation. Spend the day coding it in PySpark.
- This will test all your week's knowledge: reading data, transforms, perhaps joins, writing out results. You'll likely refer back to notes or documentation – that's fine (in fact, it reinforces learning when you look something up the second time).
- When it's working, take note of how long it took or any bottlenecks. If something was tricky, take time to understand why (e.g., data format issues, etc.).

- *Checkpoint:* By completion, you have an end-to-end Spark job running on your local data. This is a tangible achievement! Also, create a list of key Spark concepts learned this week (maybe 10-15 items) to revisit.

- **Day 7: Rest & Review** – *Duration: ~2 hours (light day).*

- Use today to rest your mind a bit – you've covered a lot in one week. Only spend a couple of hours on structured activity.
- Suggested light tasks: Go through the list of Spark concepts you wrote and see if you recall their meaning. Maybe watch a fun YouTube video about a cool Spark use case (to inspire and contextualize what you learned). If you feel up to it, do a short quiz (you can find Spark quiz questions online or make your own).
- If there were any backlog items (e.g., you planned to read a certain chapter but didn't), you can catch up today casually.
- Ensure your AWS account is set up properly for next week (check you can login, maybe create an S3 bucket in advance).
- *Checkpoint:* All core Spark fundamentals are at least 75% clear to you. You feel that you can take on using Spark in the cloud next.

Week 2: AWS Integration and Expanding Skills (Days 8–14)

- **Day 8: Intro to AWS Data Services (Theory day)** – *Duration: ~3–4 hours.*
- Transition to AWS by learning what each relevant service does. Spend the morning reading the AWS Data Engineering roadmap highlights [14] [15] and AWS service docs (especially S3, Glue, Redshift, Athena intros) to get familiar with terms.
- Afternoon: Sign in to AWS Console. Browse through S3 (create a test bucket manually), Glue (note the sections: Jobs, Crawlers, Catalog), Athena (see the query editor). No need to run anything yet, just familiarize with the interfaces.

- If you have Terraform installed, also try a basic `terraform init` with a simple config (just to ensure Terraform is ready when needed).

- *Checkpoint:* You can describe the purpose of S3, Glue, Redshift, Athena, IAM in one sentence each [52] [53]. You have AWS ready to go for hands-on starting Day 9.

- **Day 9: Hands-On AWS – S3 and Glue Crawlers** – *Duration: ~4 hours.*

- Morning (1.5h): Take a dataset (the one you used in Spark perhaps). Use AWS CLI or console to upload it to an S3 bucket (call it something like `de-training-bucket`). Ensure it's organized in a folder if needed. Then, configure a **Glue Crawler** via the console to crawl that S3 path. Run the crawler and observe the results (a table in Glue Data Catalog).
- Afternoon (2.5h): Write your first Glue ETL script. You can do this in the AWS console's script editor or locally and then upload. A simple approach: use the code from your Spark mini-project (Day 6), adjust it to run in Glue (use `GlueContext` if required, but note that standard Spark code often works by creating SparkSession in Glue as well). Create a Glue Job, assign it an IAM role (you might need to create one – use AWSGlueServiceRole managed policy and S3 access). This might be tricky – refer to AWS docs for minimal Glue IAM role policies.
- Attempt to run the Glue job on a small scale (maybe limit input to a few files to keep it quick). Monitor the job run in Glue console. If it fails, read the logs and iterate (this is normal!). Keep at it until you get a success.

- *Checkpoint:* By end of day, you have data in S3 cataloged and (ideally) a Glue job that processed data from S3 to some target (even if the target is just another S3 location). You've effectively created the skeleton of a data pipeline on AWS. Take note of any difficulties (like IAM issues, etc.) for troubleshooting tomorrow.

- **Day 10: Deeper AWS ETL – Glue to Redshift/Athena** – *Duration: ~4 hours.*

- Morning (2h): If your Day 9 Glue job was writing back to S3, now integrate Athena. Use the Glue Catalog table from the crawler or create one for the Glue job output. In Athena, run a SQL query to validate the data. This shows the data is queryable. If your plan was to load Redshift, try the Redshift step now: spin up a small Redshift (if using Redshift Serverless, it's quick). Use AWS Glue's integration to Redshift (Glue can connect to Redshift using JDBC). Maybe configure a second Glue job or modify the first to write to Redshift (could be as simple as using Redshift's `COPY` via a SQL executed in the Glue job, or using Spark's `write.format("jdbc")`). This might be advanced; if it's too time-consuming, skip the actual load and instead use the time to read how it's done or simulate to Postgres as earlier.
- Afternoon (2h): Infrastructure as Code time – take one piece of what you did and automate via Terraform. For example, write Terraform config for the S3 bucket and IAM role used by Glue. Apply it (maybe on a new bucket to avoid messing with the existing one). Verify resources created. If comfortable, also script the Glue crawler or job creation with Terraform (optional).
- This mix of activities cements both your AWS usage and Terraform intro.

- *Checkpoint:* By now, you have a functioning pipeline: data in S3 -> processed by Spark (Glue) -> accessible for analytics (Athena or Redshift). This is a **huge milestone** – essentially the outline of what many data engineering projects entail [30]. Also, you wrote some Terraform and successfully created cloud resources with it, proving you can automate deployments. Celebrate this progress!

- **Day 11: Spark + AWS Q&A and Reinforcement** – *Duration: ~3 hours.*

- After two intense hands-on days, step back and consolidate. Spend the morning reviewing what you did on AWS: draw a diagram of the pipeline, list out the AWS services used and how data flows. This reinforces the architecture in your mind.
- Make a list of issues or concepts that were confusing (perhaps IAM trust policies, or Glue vs. SparkSession stuff). Use this time to answer those. For instance, if IAM was tricky, read a specific AWS IAM tutorial; if Glue logging was hard, learn how to enable debugging in Glue jobs. This is a flex day to clear doubts.
- Afternoon: Do a Q&A with yourself or a colleague. For example: *"How would you explain the difference between EMR and Glue?"*, *"What does Athena do in our pipeline?"*, *"How do you ensure the Glue job has access to S3?"*. Answer these out loud or write them down. Use documentation to fill any gaps.

- *Checkpoint:* By end of Day 11, you should feel that the AWS part of the puzzle is making sense and you can articulate the basics. If something still feels murky, mark it for the coming days to address.

- **Day 12: Extended Project Work / Optional Learning** – *Duration: ~4 hours.*

- Use this day to extend your skills based on interest or need:
    - If you haven't fully tried Redshift, maybe devote today to that: Ingest a small table into Redshift and run a query, or read about Redshift Spectrum (Athena-like capability for Redshift to query S3 data) to mention in conversation.
    - Or, explore **Spark Streaming** concept if you think real-time might come up. You could read a tutorial on Structured Streaming and run a quick example with a socket stream on your local just to see it (only if curious and time allows).
    - Alternatively, if everything is going well, you can spend time making your project more robust: try a larger data set on your Glue job to see how it scales, or add error handling in Spark code, etc.
    - Another idea: practice writing some **PySpark unit tests** (using small functions) or using the Spark DataFrame `sample()` & `limit()` to test on subsets. This isn't required but gives you talking points on data validation.
- Essentially, Day 12 is a buffer for exploration or catching up. Use it to either go broader (touch on an adjacent tool like Kinesis or Airflow if you're interested) or deeper (improve what you built).

- *Checkpoint:* If you had any lingering "optional" topics you wanted to peek at, you've done so. Importantly, ensure by this day that no critical topic remains completely untouched.

- **Day 13: Terraform & DevOps Focus** – *Duration: ~3 hours.*

- Focus entirely on Terraform and infrastructure practices today.
- Morning: Take the Medium tutorials on Terraform for Data Engineers [43] [17] and follow one end-to-end (for example, creating the data lake infra with Glue crawler and Athena). Do this in a controlled way: maybe use a different AWS account or be careful to not delete your existing resources. The goal is to see a full Terraform example and run it.
- Afternoon: Once done, analyze the Terraform scripts you wrote or used. Identify where you can parameterize (use variables) or output values. Maybe refactor your previous Terraform code with knowledge gained. Also consider writing a Terraform plan for something new: e.g., "If I had to deploy this whole pipeline to a new environment, what would I code?" – outline the resources in Terraform (even if you don't execute it due to time).
- If time permits, also think about CI/CD: how would you integrate Terraform in a pipeline? (e.g., using Terraform Cloud or GitHub actions – you can just read an article about it).

- *Checkpoint:* By end of Day 13, you should have no fear of Terraform. You may not be an expert, but you know how to approach writing config for AWS resources and you've done it for key components (S3, IAM, possibly Glue). You're also aware of the importance of DevOps in data engineering and can mention something about how you'd automate deployments.

- **Day 14: Week 2 Review and Rest** – *Duration: ~2–3 hours, light.*

- It's review day for AWS & Terraform content. Similar to Day 7, keep it light to consolidate learning and rest up.
- Tasks: Review the AWS architecture diagram and explain it one more time to ensure clarity. Go through your notes from Week 2 – is there any acronym or service you keep mixing up? Clarify it now.
- Do a final run of your pipeline if possible – from S3 ingest to Athena query – just to relish that it works and to imprint the flow in memory. Maybe try to optimize something minor, like turning on compression on output, just for a final bit of learning.
- Write down any key AWS limits or considerations you picked up (e.g., "Glue max memory is X", "Athena charges by TB scanned – partition your data"). These are golden nuggets to mention in discussions to show you didn't just do it, you understood the context.
- *Checkpoint:* You have completed two weeks of intensive learning! At this point, you're essentially prepared in all target areas (Spark, AWS, Terraform). Give yourself a pat on the back. You're now moving to final polishing and confidence-building.

Week 3: Refinement, Project Completion, and Assessment Readiness (Days 15–20)

- **Day 15: Full Pipeline Test & Documentation** – *Duration: ~3 hours.*
- Today, treat your project as if it were going to production (or at least a demo). Run through the entire process end-to-end without interruption: from getting the raw data (you might simulate "new data arrives" by copying a file to S3) to running the ETL (Spark/Glue job) to querying the result. Do this as one smooth workflow, possibly timing it or writing a script to kick off each part (if ambitious, you could even write a small shell script or use AWS Step Functions – but only if you have time and interest).
- As it runs, note if any part is flaky or slow. Improve it if feasible (maybe your Glue job needs more workers for speed, or you add an index in Redshift if queries are slow – small optimizations to think about).
- After confirming everything works, focus on **documentation**: Write a brief summary of the project: purpose, tools, steps, and outcomes. This is partly for you to solidify understanding, and partly to have something you could show or refer to in your assessment. Even if the assessors don't explicitly ask, you having a clear mental (or written) picture means you can confidently bring it up: *"I actually practiced by building a mini data pipeline with Spark on AWS – let me walk you through it."* That could be impressive.
- If documentation is done quickly, use extra time to tidy up: organize your code repository, perhaps push it to GitHub (if allowed and minus any sensitive info) or at least zip it for sharing. A well-documented project can act as a portfolio piece.

- *Checkpoint:* Your project is essentially done and documented. It's a concrete evidence of your skills now. You're also verifying that each piece of technology is working in concert, which is great rehearsal for explaining integration.

- **Day 16: Mock Assessment and Q&A Practice** – *Duration: ~3–4 hours.*

- Put yourself in the hot seat. Simulate an assessment by preparing a list of likely questions or tasks and then addressing them. Potential areas to cover:
    - **Spark theory:** e.g., "Explain how Spark achieves fault tolerance" (you'd mention RDD lineage/DAG recomputation) or "What's the difference between an RDD and a DataFrame?"
    - **Code reading/writing:** maybe take a snippet of PySpark code (from a blog or your own) and explain what it does, as if the interviewer asked you to interpret code. Or write a short piece of pseudocode on the spot (like, "How would you use Spark to count occurrences of words in a large dataset?" – then you outline the steps).
    - **AWS scenario:** e.g., "Suppose we have a terabyte of data arriving daily, how would you design a pipeline for that on AWS?" – you can answer with what you know: ingest to S3, use Glue/Spark to process in parts, store in Redshift or query via Athena, etc., discussing partitioning and scaling.
    - **Terraform/DevOps:** e.g., "Why use Terraform in data engineering?" or "How would you manage different environments (dev/test/prod) for your data pipeline?" – you could answer about Terraform workspaces or separate configs, and CI/CD pipelines deploying to each environment, etc.
    - **General data engineering:** maybe a question on data partitioning strategy, or how to handle schema changes – drawing on your fundamentals book knowledge. Also, your BI background might prompt a question like "How is designing a data warehouse ETL different or similar to a Spark data pipeline?" – be ready to highlight differences (e.g., SQL vs. code, batch vs. maybe incremental, etc.).
- To make it effective, **speak your answers out loud** as if in an interview. If possible, record yourself. Then listen and critique: did you use filler words, did you structure the answer clearly, did you hit the key points? This will improve your communication.
- If you have a willing friend/colleague, do this mock Q&A with them acting as interviewer. Even a non-tech friend can help by reading your prepared questions and saying "I didn't understand when you said X" which is valuable feedback on clarity.

- *Checkpoint:* After this, you should feel a lot more comfortable with the idea of the assessment. Any question that stumped you in mock, research and resolve it now. It's better to be stumped in practice than in the real thing. This also builds confidence – you'll likely realize you actually can answer most questions now, which is reassuring.

- **Day 17: Last Content Review & Weak Spot Fixing** – *Duration: ~3 hours.*

- By day 17, you have identified any remaining weak areas from the mock assessment or personal feeling. Use today to **target those specifically**. For example: if you feel unsure about Spark streaming or some advanced Spark optimization, read a concise tutorial or watch a talk on it. If IAM policies were still confusing, draw a diagram or read AWS docs to clarify. If you haven't tried a particular feature you think might come up (like Spark MLlib, or a Glue feature), at least familiarize yourself with it conceptually.
- Go through your flashcards or notes one final time for any definitions or facts you want to keep straight (like "Spark default partition count is based on HDFS block size... or Glue job max 100 DPUs..." small facts can sometimes impress, but don't overload on them).
- Ensure you also review any **fundamental BI/DE concepts** that could be asked generally: e.g., data normalization vs. denormalization (since you have DW background, maybe they expect you to know schema design), or ACID vs eventual consistency (since dealing with distributed storage). Again, these are broad, but a quick refresh can't hurt.

- *Health check:* Also take a moment to check your own wellbeing – the last days have been intense. Make sure you're getting enough rest, and plan something relaxing later today or tomorrow. Sometimes stepping away helps consolidate memory, so don't hesitate to take a breather.

- *Checkpoint:* No glaring weak spot remains unaddressed. You have a solid answer or approach ready for all major expected topics.

- **Day 18: Light Practice and Rest** – *Duration: ~2 hours.*

- You're very close now. Use Day 18 to do a light practice just to keep your mind engaged, but avoid heavy studying to prevent burnout. For instance:
    - Spend an hour on an interactive coding challenge or Kaggle Kernels – something fun but related, like implementing a small Spark job for a challenge problem, or reading an interesting article on cutting-edge Spark (to get inspired).
    - If you enjoy writing, perhaps write a short blog-style entry about what you learned in these weeks. Teaching others is a great way to reinforce knowledge one last time.
- Make sure all your materials are organized for quick refreshing. Perhaps bookmark key references in your browser in case you want to glance on Day 19/20.

- *Checkpoint:* You are mentally recharging. Everything important is already in your head; trust that. Day 18 is about keeping your skills warm without overexertion.

- **Day 19: Final Revision (Hit the High Notes)** – *Duration: ~3 hours.*

- This is your final active study day. Use it to cement key points one last time:
    - Skim through your summary notes for Spark, AWS, Terraform. Don't deep dive, just recall each point and ensure it triggers understanding.
    - Do a final run of any demonstration you might give. For example, if during the assessment you plan to bring up your mini-project, rehearse the 2-minute version of explaining it: (problem -> solution -> tech used -> outcome). This storytelling ability will make you stand out.
    - If there are any factoids or terminology you keep forgetting, memorize them now (e.g., "What's the term for Spark's optimization engine? Catalyst.", "Name an AWS service for real-time streaming: Kinesis." etc.). It's easier to remember a few last facts now and they might be useful in conversation.
- Keep stress low: this is polishing, not learning new things. Everything critical was done by Day 17.

- *Checkpoint:* By the end of Day 19, you should feel a sense of completion. You've done everything you could within reason. You are able to explain and perform the tasks that were set out. Any remaining uncertainties are minor and unlikely to trip you up significantly.

- **Day 20: Assessment Day (or Simulation)** –

- If the actual assessment or start of your role is today, you're ready. If not, treat today as the day to simulate that final step. Wake up with the mindset that you are now a data engineer about to demonstrate your skill.
- Since heavy studying today won't help much, do something light in the morning: maybe a brief glance at notes or just relax. Confidence is key now – you want to be in a positive headspace.
- When it's time for the assessment or a final self-evaluation, **present your knowledge confidently**. You have a strong foundation now: you can reason through Spark problems, talk

about AWS data workflows, and mention Terraform when infrastructure comes up, all backed by hands-on experience.

- *Post-assessment:* Reflect on how it went, and note if there were any questions you felt weak on – this can guide a bit of follow-up learning. But hopefully, by following this rigorous plan, you will find that you were well-prepared and the questions were mostly on things you had already practiced or reviewed, leading to a successful evaluation.

This detailed schedule ensures that each day has a clear focus, preventing last-minute cramming and balancing intense learning with necessary rest. It aligns the learning tasks with the time available, ensuring you cover all bases (Spark, AWS, Terraform, etc.) without overloading any single day. By sticking to this schedule, you can be confident that in 20 days you will have transformed from a Spark beginner to someone ready to join the data engineering practice, with a robust understanding of PySpark on AWS and the ability to apply it in a project – exactly what you set out to achieve. Good luck, and enjoy the learning journey! [14] [15]

---

[1] [2] [3] [6] [50] [51] PySpark for AWS Glue: A Comprehensive Guide to Big Data Processing | by Shuvradeep Ghosh | Medium
https://medium.com/@shuvradipghosh97/pyspark-for-aws-glue-a-comprehensive-guide-to-big-data-processing-407f921d3452

[4] [5] [7] [8] [9] [10] [11] [48] [49] Guide | Apache Spark architecture: Concepts, components, and best practices
https://www.instaclustr.com/education/apache-spark/apache-spark-architecture-concepts-components-and-best-practices/

[12] [19] [20] [21] [42] AWS Data Engineering Roadmap: Skills and Steps to Success | by Alejandro Ibáñez Peña | Medium
https://medium.com/@alejandro.ibapen/aws-data-engineering-roadmap-skills-and-steps-to-success-e2484a8dfb99

[13] [24] [34] [35] [36] [37] How to Learn Apache spark? : r/dataengineering
https://www.reddit.com/r/dataengineering/comments/t0ymzm/how_to_learn_apache_spark/

[14] [15] [16] [52] [53] From Zero to Hero: Data Engineering on AWS for Beginners
https://dataengineeracademy.com/blog/from-zero-to-hero-data-engineering-on-aws-for-beginners/

[17] [18] [27] [28] [29] [32] [44] Using Terraform To Create a Data Lake and Serverless Data Analytic Infrastructure on AWS: Glue, S3, IAM, Athena | by Data Dev Backyard | AWS Tip
https://awstip.com/using-terraform-to-create-a-data-lake-and-serverless-data-analytic-infrastructure-on-aws-glue-s3-cdf0e450654e?gi=7e6370bf3cc6

[22] [23] [31] [43] Terraform for Data Engineers. In this tutorial series, I would like… | by Data Dev Backyard | Medium
https://medium.com/@data.dev.backyard/terraform-for-data-engineers-1f364f645954

[25] [26] [30] [33] [47] Streamline ETL: S3 to Redshift Serverless with AWS Glue | Data Engineer Things
https://blog.dataengineerthings.org/how-to-build-a-data-pipeline-with-aws-glue-and-terraform-ac1ace165d29?gi=860360e11931

[38] Getting Started with AWS Glue: A Step-by-Step Guide | DataCamp
https://www.datacamp.com/tutorial/aws-glue

[39] PySpark For AWS Glue Tutorial [FULL COURSE in 100min] - YouTube
https://www.youtube.com/watch?v=DICsZiwuHJo

[40] AWS Glue - Pluralsight
https://www.pluralsight.com/professional-services/cloud/aws-glue

41 Build an ETL service pipeline to load data incrementally from ...
https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/build-an-etl-service-pipeline-to-load-data-incrementally-from-amazon-s3-to-amazon-redshift-using-aws-glue.html

45 Automate S3 Data ETL Pipeline With AWS Glue to Read ... - YouTube
https://www.youtube.com/watch?v=rISNU8ZmG64

46 Mastering AWS Data Analytics with Terraform and GitLab - Medium
https://medium.com/@williamwarley/mastering-aws-data-analytics-with-terraform-and-gitlab-a-comprehensive-guide-cd92029afacd