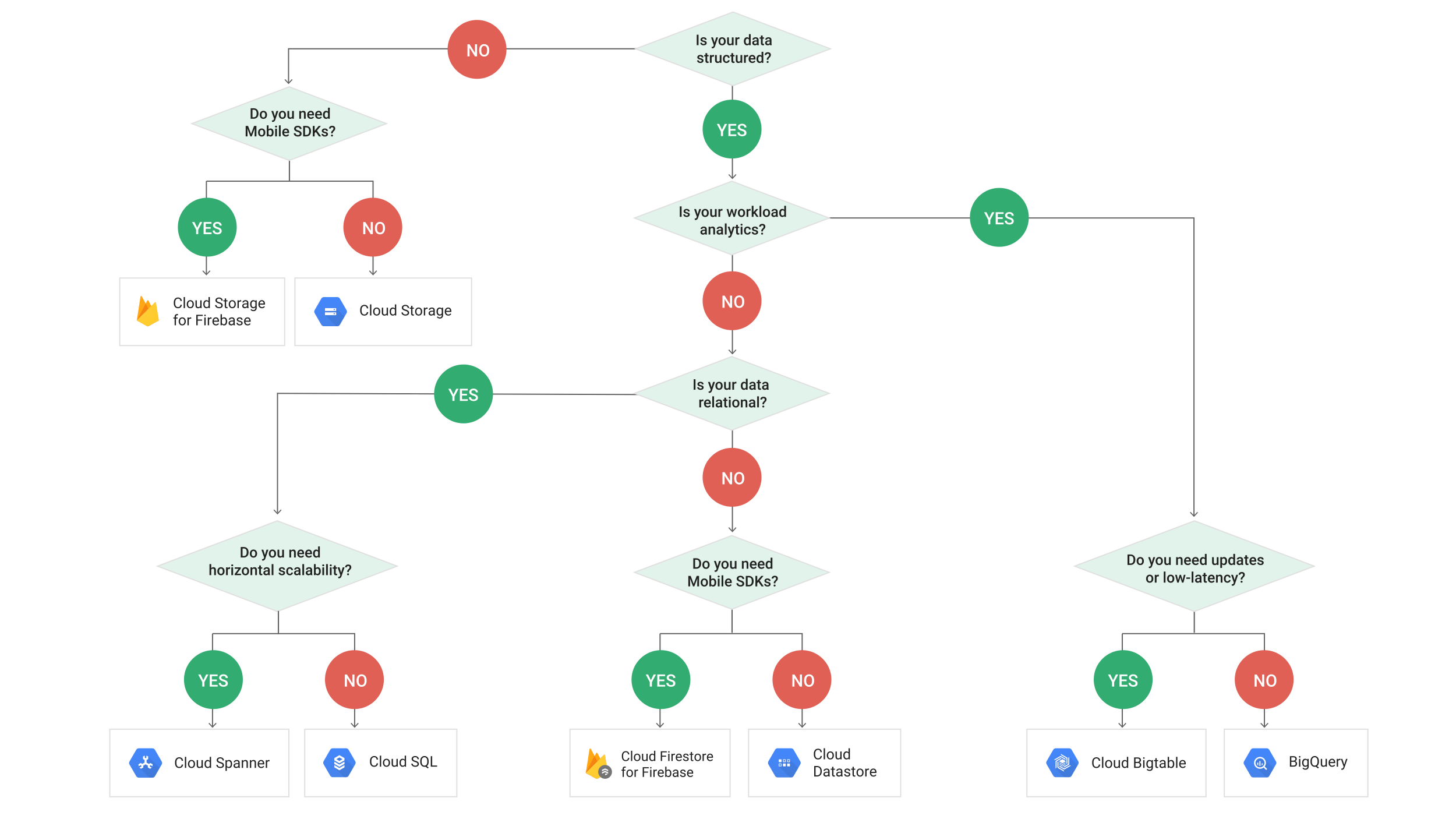
**Google Cloud Data Engineer Certificate Study Guide**

**Exam Overview**

* Storage (20%)
  + GCS, Cloud SQL, DataStore, BigTable, BigQuery
* Big Data Processing (35%)
  + BigQuery, Dataflow, Dataproc, Datalab, Pub/Sub
* Machine Learning (18%)
  + ML APIs, TensorFlow
* Case Studies (15%)
* Others (Hadoop and Security about 12%)

**Storage**



**Google Cloud Storage (GCS)**

* Blob storage. Content not indexed.
* Virtually unlimited storage.
* Can have domain name buckets
* Can make requesters pay (ex. requester in different project)
* Pub/Sub can have notifications based on operations to buckets/objects
* Objects are immutable
* Can set Cache-Control metadata for frequently accessed objects
* Keep in mind compliance requirements when storing data in certain regions.
* No native directory support
  + Forward slashes have no special meaning
  + Performance of a native filesystem is not present.
* Storage classes can change, but the objects (files) within them retain their storage class.
* Not ideal for high volume read/write
* A way to store data that can be commonly used by Dataproc and Bigquery
* **Storage Classes**
  + **Multi-regional**
    - Serving website content, interactive workloads, mobile game/gaming applications
    - Highest availability
    - Geo-redundant: Stores data in at least 2 regions separated by at least 100 miles within the multi-regional location of the bucket.
  + **Regional**
    - Storing data used by Compute Engine
    - Better performance for data-intensive computation
  + **Nearline**
    - Accessed once a month max
    - 30 day min. storage duration
    - Ex. Data backup, disaster recovery, archival storage
  + **Coldline**
    - Accessed once a year max
    - 90 day min. storage duration
    - Ex. Data stored for legal or regulatory reasons
* **Versioning**
  + Needs to be enabled
  + Things this enables:
    - List archived versions of an object
    - Restore live version of an object from an older state
    - Permanently delete an archived version
  + Archived versions retain ACLs and does not necessarily have same permissions as live version of object.
* **Encryption**
  + **Encryption at rest (Google-Managed Encryption Keys)**
    - Default (AES-256)
    - Use TLS or HTTPS to protect data as it travels over Internet
  + **Server-side encryption:**
    - Layers on top of default encryption
    - Occurs after GCS receives data, but before written to disk
      * **Customer-supplied encryption keys**
        + Provide key for each GCS operation
        + Key purged from servers after operation is complete
        + Stores only a cryptographic hash of key for future requests
        + Transfer Service, Dataflow, and Dataproc do not support this currently
        + Key rotation

Edit .boto config file

Encryption\_key = [NEW\_KEY]

Decryption\_key1 = [OLD\_KEY]

gsutil rewrite -k gs:://[BUCKET]/[OBJECT]

* + - * **Customer-managed encryption keys**
        + Generate and manage keys using Cloud Key Management Service (KMS)
        + KMS can be independent from the project that contains buckets (separation of duties)
        + Uses service accounts to encrypt/decrypt
        + Cloud SQL exports to GCS and Dataflow do not support this currently
  + **Client-side encryption:**
    - Occurs before data sent to GCS
    - GCS performs default encryption on it as well.
* **Storage Transfer Service**
  + Transfers data from an online data source (Amazon S3, HTTP/HTTPS location, GCS bucket) to a data sink (always GCS bucket).
  + Use cases:
    - Backup data to GCS from other storage providers
    - Move data from one GCS bucket to another (enables availability to different groups of users or applications)
    - Periodically move data as part of a processing pipeline or analytical workflow
  + Schedule one-time transfer operations or recurring ones
  + Delete existing objects in the destination bucket if they don’t have a corresponding object in source
  + Delete source objects after transferring them
  + Schedule periodic synchronization from data source to data sink with advanced filters based on file creation data, file-name filters, and the times of day you prefer to import data.
  + **Transfer Service vs. Gsutil**
    - On premise data source : gsutil
    - Another cloud storage provider data source : Transfer Service

**Cloud SQL**

* Managed/No ops relational database (PostgreSQL, MySQL)
  + Complex queries perform better in postgresq
* Best for **gigabytes** of data with **transactional** nature
  + Low latency
  + Doesn’t scale well beyond GB’s
  + Data structures and underlying infrastructure required
* Too slow for analytics/BI/warehousing (OLAP)
* 2nd Generation Allow
  + Cloud Proxy Support
  + Higher availability configurations
  + Maintenance won’t take down the server
* Use SSD for production (instead of hard disk (persistent disk))
* Enable binary logging
  + For Point-in-time recovery and replication
* Bulk Loading Data
  + Copy data to to GCS
  + Import it into DB using copy from csv or something similar.

**Cloud Spanner**

* Distributed and scalable solution for RDBMS (more expensive)
* Horizontal scaling: Add more machines
* Use when:
  + Need high availability
  + Strong consistency
  + Transactional support for reads and writes (especially writes)
* Don’t use when:
  + Data is not relational, or not even structured
  + Want an open source RDBMS
  + Strong consistency/availability is overkill
* **Data Model**
  + Specifies a parent-child relationship for efficient storage
  + Interleaved representation (like **HBase**)
* **Parent Child Relationship**
  + Between tables
  + Cause physical location for fast access
    - i.e. query Students and Grades together, make Grades child of Student
  + Primary key of parent table **must** to be part of the key in the interleaved child table.
* **Interleaving**
  + Rows are stored in sorted order of primary key values
  + Child rows are inserted between parent rows with that key prefix
* **Hotspotting**
  + Need to choose primary keys carefully (like **HBase**)
  + Do not use monotonically increasing values, else writes will be on the same locations.
  + Use hash of key value if using naturally monotonically ordered keys (serial in postgres)
* **Splits**
  + Parent-child relationship can get complicated (i.e. 7 layers deep)
  + Spanner is distributed – uses “splits”
  + Split – Range of rows that can be moved around independent of other rows
  + Added to distribute high read-write data (to break up hotspots)
* **Secondary Indices**
  + Key-based storage ensures fast sequential scan of keys (like **HBase**)
  + Can also add secondary indices (**unlike HBase**)
    - Can cause data to be stored twice
      * i.e. Grades -> Course table | Grades -> Students table
  + Fine grained control on use of indices
    - Force query to use specific index: **Index Directives**
    - Force column to be copied into secondary index (use a STORING clause)
* Data Types
  + Non-normalized types such as ARRAY and STRUCT available too.
    - STRUCTs: NOT OK in tables, but can be returned in queries
    - ARRAYs: OK in tables, but ARRAYs of ARRAYs are not
* Transactions
  + Supports serializability
    - All transactions appear if they were executed in a serial order, even if some operations of distinct transactions actually occurred in parallel.
  + Stronger than traditional ACID
    - Transactions commit in an order that is reflected in their commit timestamps
    - Commit timestamps are “real time”
  + 2 Transaction Modes
    - Locking read-write
      * Slow
      * Only one that supports writing data
    - Read-only
      * Fast
      * Only requires read locking
  + If making a one-off read use “**Single Read Call**”
    - Fastest, no transaction checks needed!
* Staleness
  + Can set timestamp bounds
    - Strong: Read latest data
    - Bounded Staleness: Read version no later than …
      * Could be in past or future
* Production Environment
  + At least 3 nodes
  + Best performance when each CPU is under 75% utilization

**DataStore**

**BigTable**

**BigQuery**

**Dataflow**

**Dataproc**

**Datalab**

**Pub/Sub**

**ML APIs**

**TensorFlow**

**Hadoop**

**Security**