NoSQL Databases

Cloud Datastore and Firestore

Cloud Datastore

- **Datastore** - Highly scalable NoSQL Document Database
  - Automatically scales and partitions data as it grows
  - Recommended for upto a few TBs of data
    - For bigger volumes, BigTable is recommended
  - Supports Transactions, Indexes and SQL like queries (GQL)
    - Does NOT support Joins or Aggregate (sum or count) operations
  - For use cases needing flexible schema with transactions
    - Examples: User Profile and Product Catalogs
  - Structure: Kind > Entity (Use namespaces to group entities)
  - You can export data ONLY from gcloud (NOT from cloud console)
    - Export contains a metadata file and a folder with the data
- **Firestore** = Datastore++ : Optimized for multi device access
  - Offline mode and data synchronization across multiple devices - mobile, IOT etc
  - Provides client side libraries - Web, iOS, Android and more
  - Offers Datastore and Native modes

Understanding Cloud Datastore Best Practices

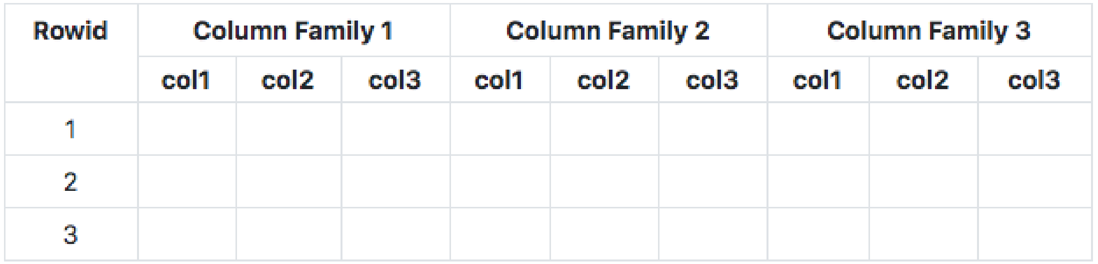
Cloud Datastore

- Cloud Datastore is a **document store with flexible schema**
  - Recommended for storing things like user profiles
  - Another Use Case: Index for objects stored in Cloud Storage
    - You want to allow users to upload their profile pictures:
      - Store objects (pictures) in Cloud Storage
      - Enable quick search by storing metadata (like ids and cloud storage bucket, object details) in Cloud Datastore
- Design **your keys and indexes** carefully:
  - Avoid monotonically increasing values as keys
    - NOT RECOMMENDED - 1, 2, 3, …, OR "Customer1", "Customer2", "Customer3", … or timestamps
    - RECOMMENDED - Use allocateIds() for well-distributed numeric IDs
  - Create indexes only if they would be used in queries
    - For ad hoc queries on large datasets without pre-defined indexes, BigQuery is recommended!
- **Prefer batch operations** (to single read, write or delete operations):
  - More efficient as multiple operations are performed with same overhead as one operation

Cloud BigTable

- **Petabyte scale, wide column** NoSQL DB (HBase API compatible)
  - Designed for huge volumes of analytical and operational data
    - IOT Streams, Analytics, Time Series Data etc
  - Handle millions of read/write TPS at very low latency
  - Single row transactions (multi row transactions NOT supported)
- **NOT serverless**: You need to create a server instance (Use SSD or HDD)
  - Scale horizontally with multiple nodes (No downtime for cluster resizing)
- **CANNOT export data using cloud console or gcloud**:
  - Either use a Java application (java -jar JAR export\import) OR
  - Use HBase commands
- Use cbt command line tool to work with BigTable (NOT gcloud)
  - Ex: **cbt createtable my-table**

Cloud BigTable - Wide Column Database



Rowid	Column Family 1			Column Family 2			Column Family 3		
	col1	col2	col3	col1	col2	col3	col1	col2	col3
1									
2									
3									

- At the most basic level, each table is a sorted key/value map
  - Each value in a row is indexed using a key - **row key**
  - Related columns are grouped into column families
    - Each column is identified by using column-family:column-qualifer(or name)
- This structure supports high read and write throughput at low latency
  - **Advantages** : Scalable to **petabytes of data** with **millisecond responses** upto **millions of TPS**
- **Use cases** : IOT streams, graph data and real time analytics (time-series data, financial data - transaction histories, stock prices etc)
- **Cloud Dataflow** : Used to export data from BigTable to CloudStorage

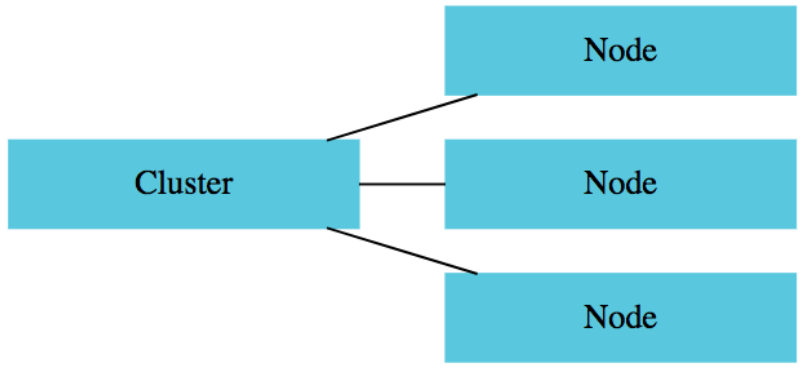
Designing BigTable Tables

- **Two things** you should know before starting with Bigtable:
  - What data do you want to store? (format, columns etc)
  - What would your frequently used queries look like (ranked by usage)?
- Design your table: Cloud Bigtable is a **key/value store**
  - Each table has **ONLY ONE** index, the row key
  - **Design your row key** based on your frequently used queries
    - You can have multiple row key segments - Separated by a delimiter (ex: ranga#123456#abcd)
    - Avoid sequential row keys (timestamps or sequential numbers)
      - Include timestamp as part of your row key IF you plan to retrieve data based on the timestamp
      - Use reversed timestamp (Ex: Long.MAX_VALUE - timestamp) if you frequently query recent data
        - Records will be ordered from most recent to least recent
  - After your design your table:
    - Test (heavy load for several minutes + one hour simulation) with atleast 30 GB of test data
    - Analyze usage patterns with **Key Visualizer tool** for Cloud Bigtable

Cloud Bigtable

Understanding Cloud BigTable Best Practices

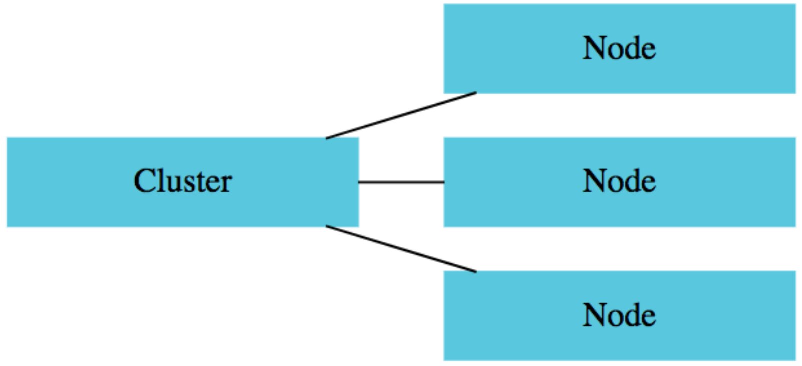
- Recommended for streaming IOT & time series data
- **Automatically shards data** into multiple tablets across nodes in cluster:
  - **Goal 1**: Have same amount of data on each node
  - **Goal 2**: Distribute reads and writes equally across all nodes
  - (REMEMBER) Pre-test with heavy load for a few minutes before you run your tests
    - Gives Bigtable a chance to balance data across your nodes
- Cloud Bigtable supports SSD or HDD storage:
  - **SSD** - For most usecases
  - **HDD** - For large non latency-sensitive data sets of size >10 TB with very very few reads



Understanding Cloud BigTable Best Practices - Replication

In28 Minutes

- You can create a Cloud Bigtable instance with **more than one cluster** to enable **replication (Cross Region or Cross Zone)** :
  - Independent copy of data is stored in each cluster (in the zone of the cluster)
  - Bigtable automatically replicates changes
  - Replication **improves durability and availability** of your data
    - Stores separate copies in multiple zones or regions
    - Can automatically failover between clusters if needed
  - Replication helps you to **put data closer to your customers**
    - Configure an application profile, or app profile with routing policy of multi-cluster routing
      - Automatically route to nearest cluster in an instance


Node
Cluster
Node
Node