GOOGLE CLOUD PLATFORM

CLOUD BIGTABLE

WHAT IS CLOUD BIGTABLE

Description

- A scalable, fully-managed NoSQL wide-column database that is suitable for both real-time access and analytics workloads.

Good for

- Low-latency read/write access
- High-throughput analytics
- Native time series support

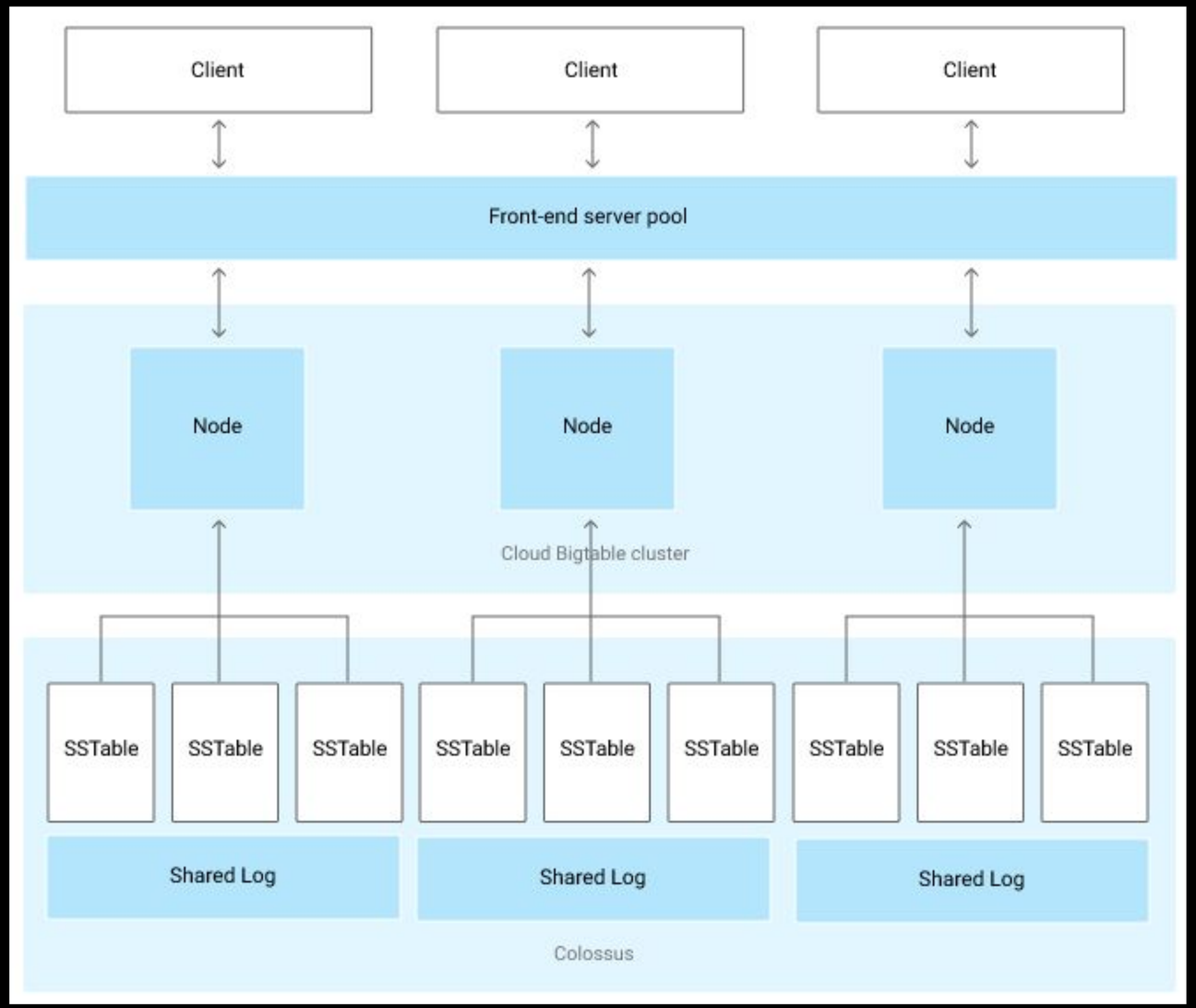
Common Workloads

- IoT, finance, adtech
- Personalization, recommendations
- Monitoring
- Geospatial datasets
- Graphs

FEATURES

- Cloud Bigtable is ideal for storing very large amounts of single-keyed data with very low latency.
- It supports high read and write throughput at low latency, and it is an ideal data source for MapReduce operations.
- Cloud Bigtable compresses your data automatically using an intelligent algorithm.
- Cloud Bigtable's powerful back-end servers offer several key advantages over a self-managed HBase installation:
  - **Incredible scalability**
  - **Simple administration**
  - **Cluster resizing without downtime**

ARCHITECTURE



- Metrics you can monitor:

CPU: Average utilization, utilization of the hottest node

Disk: Load, Storage utilization %/bytes

- Cloud Bigtable instances with only 1 cluster do not use replication

ARCHITECTURE

- Client requests go through a front-end server before they are sent to a Cloud Bigtable node.
- Bigtable nodes are separate from the actual storage
- Multiple nodes map to data based on the metadata
- A Cloud Bigtable table is sharded into blocks of contiguous rows, called *tablets*, to help balance the workload of queries.
- Rebalancing tablets from one node to another is very fast, because the actual data is not copied.
- When a Cloud Bigtable node fails, no data is lost.
- Throughput scales linearly – nodes add approximately 10,000 queries per second

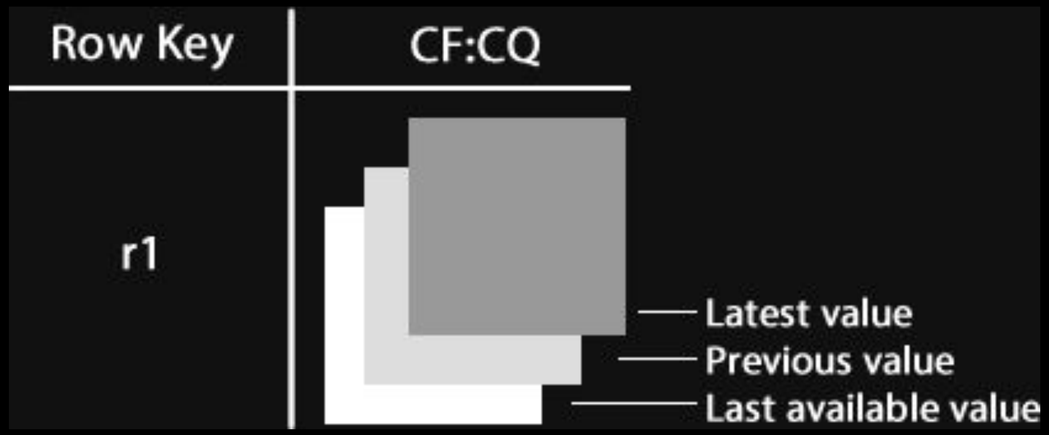
DATA MODEL

- NoSQL, no-joins, distributed key-value store
- A single value in each row called the row key is indexed
- Has a family column and
- Supports atomic single-row transactions
- Unwritten cells do not take up space

	Column-Family-1		Column-Family-2	
Row Key	**Column-Qualifier-1**	**Column-Qualifier-2**	**Column-Qualifier-1**	**Column-Qualifier-2**
r1	r1, cf1:cq1	r1, cf1:cq2	r1, cf2:cq1	r1, cf2:cq2
r2	r2, cf1:cq1	r2, cf1:cq2	r2, cf2:cq1	r2, cf2:cq2

CELLS

- Every cell is versioned with a timestamp by default
- Garbage collection retains the latest version
- Expiration can be set at the column-family level
- Periodic compaction reclaims unused space from cells
- Store items related to a given entity in a single row and related entities in adjacent rows



TIME SERIES DATA

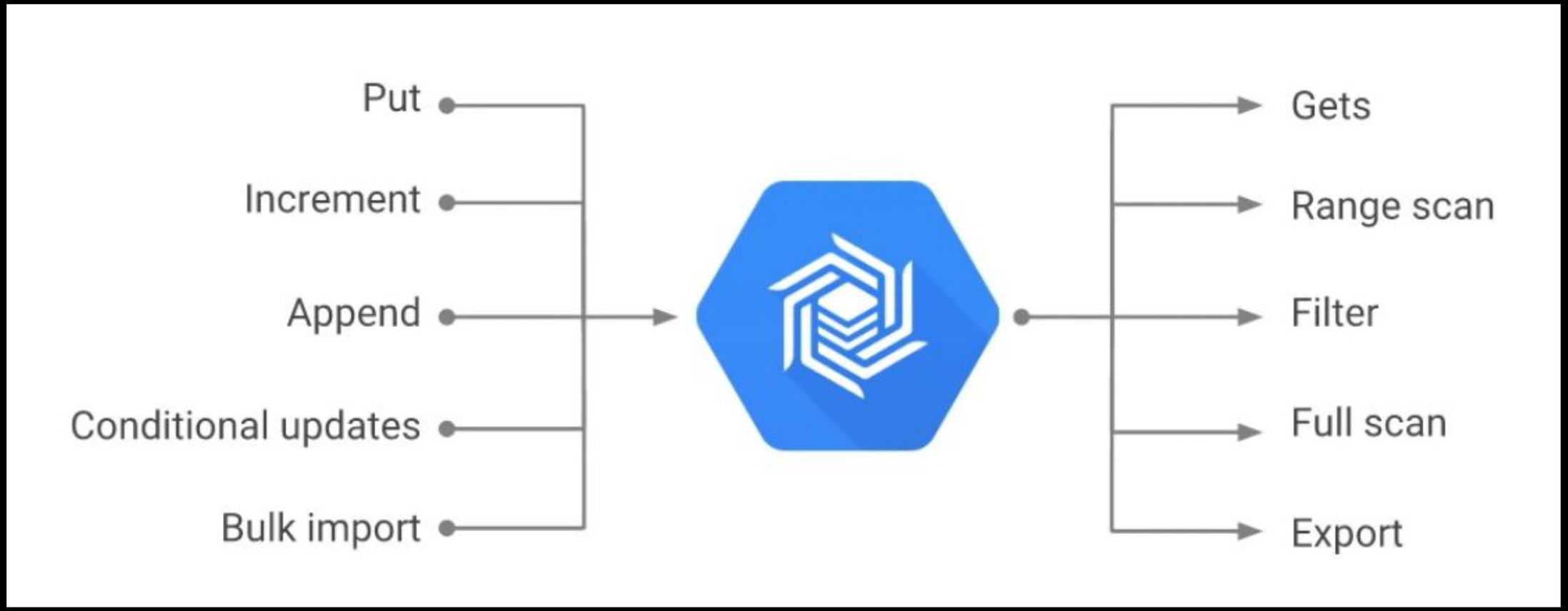
- Time series data – You record time together with a measurement
- Three design patterns for storing time-series data:
  - General patterns
    - Keep names short but meaningful – e.g. Cell vs CellPhoneNumber
  - Patterns for row key design
    - Use tall and narrow tables
    - Prefer rows to column versions
    - Design your row key with your queries in mind
    - **Solutions:** Denormalization, Query and filter
    - Ensure that your row key avoids hotspotting
    - **Solutions:** Field promotion, Salting
    - Reverse timestamps only when necessary

TIME SERIES DATA

- **Time series data –** You record the time together with a measurement
- Three design patterns for storing time-series data:
  - Patterns for data column design
    - Rows can be big but are not infinite- keep row sizes below 100 MB and column values below 10 MB
    - Keep related data in the same table, keep unrelated data in different tables
    - Store data you will access in a single query in a single column family
    - Don't exploit atomicity of single rows

INPUT OUTPUT

- **Uses load balancer as a proxy for connections**



DEMO: CREATE BIGTABLE INSTANCE

1. Navigate to Storage > Bigtable
2. Specify *instance type* (production or development)
3. Specify *storage type* (SSD or HDD)
4. Choose *application profiles*, for instances that use replication

DEMO: MANAGING TABLES

Creating a table
cbt createtable [TABLE_NAME]

Adding column families
cbt createfamily [TABLE_NAME] [FAMILY_NAME]

Deleting column families
cbt deletefamily [TABLE_NAME] [FAMILY_NAME]

Viewing information about a table
cbt ls [TABLE_NAME]

Setting garbage collection policies
cbt setgcpolicy [TABLE_NAME] [FAMILY_NAME] maxversions=[VERSIONS]

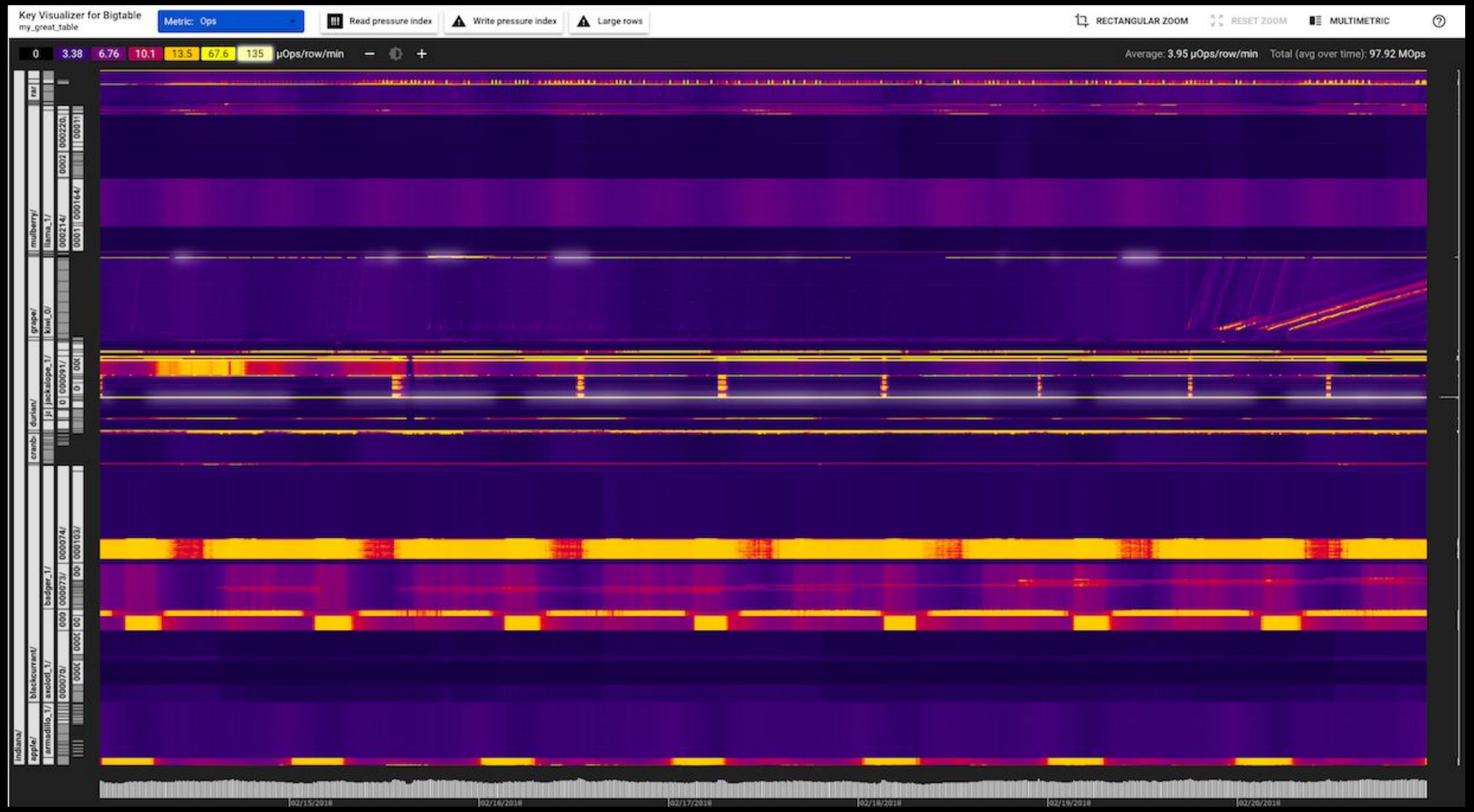
REPLICATION

- To use replication in a Cloud Bigtable instance, create a new instance with 2 clusters, or add a second cluster to an existing instance.
- Cloud Bigtable replicates the following types of changes automatically:
  - Updates to the data in existing tables
  - New and deleted tables
  - Added and removed column families
  - Changes to a column family's garbage-collection policy
- Consistency model
  - eventually consistent
  - read-your-writes consistency
  - strong consistency
- If a Cloud Bigtable cluster becomes unresponsive, replication makes it possible for incoming traffic to fail over to the instance's other cluster.

KEY VISUALIZER

- Generates visual reports for your tables that break down your usage based on the row keys that you access.
- Key Visualizer can help you complete the following tasks:
  - Check whether your reads or writes are creating hotspots on specific rows
  - Find rows that contain too much data
  - Look at whether your access patterns are balanced across all of the rows in a table
- Key Visualizer automatically generates hourly and daily scans for every table where the table size was over 30 GB or the average of all reads or all writes was at least 10,000 rows per second in the last 24 hours
- Each scan includes a large heatmap, which shows access patterns for a group of row keys over time and the aggregate values

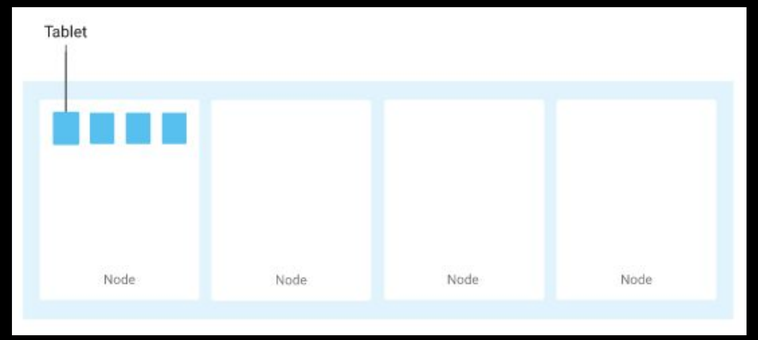
KEY VISUALIZER - HEATMAP



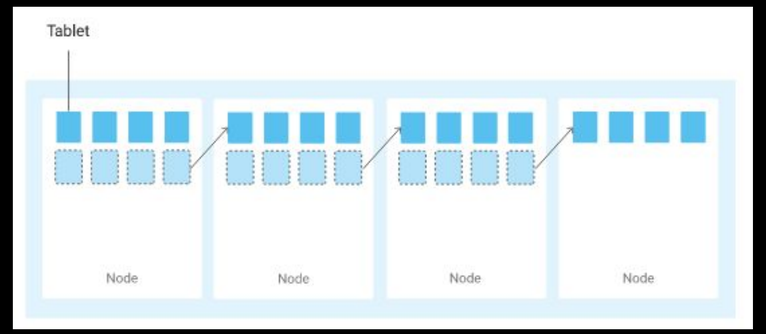
PERFORMANCE

- A cluster's performance increases linearly as you add nodes to the cluster.
- Cloud Bigtable shards the data into multiple tablets, which can be moved between nodes in your Cloud Bigtable cluster.

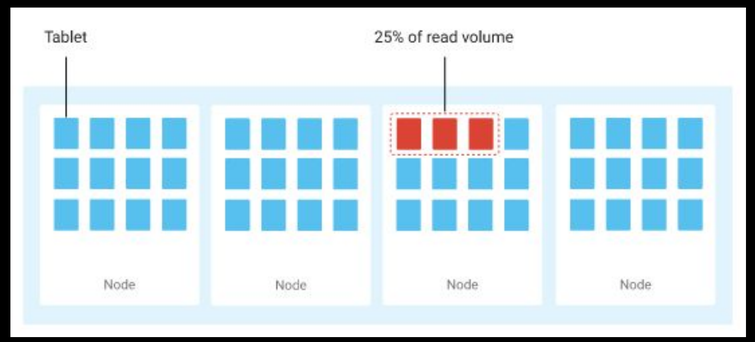
1.



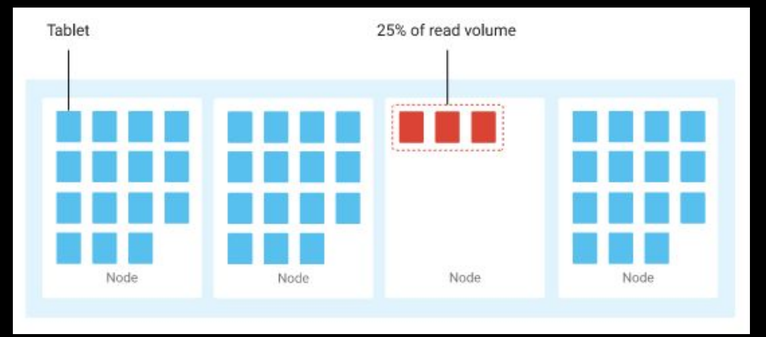
2.



3.



4.

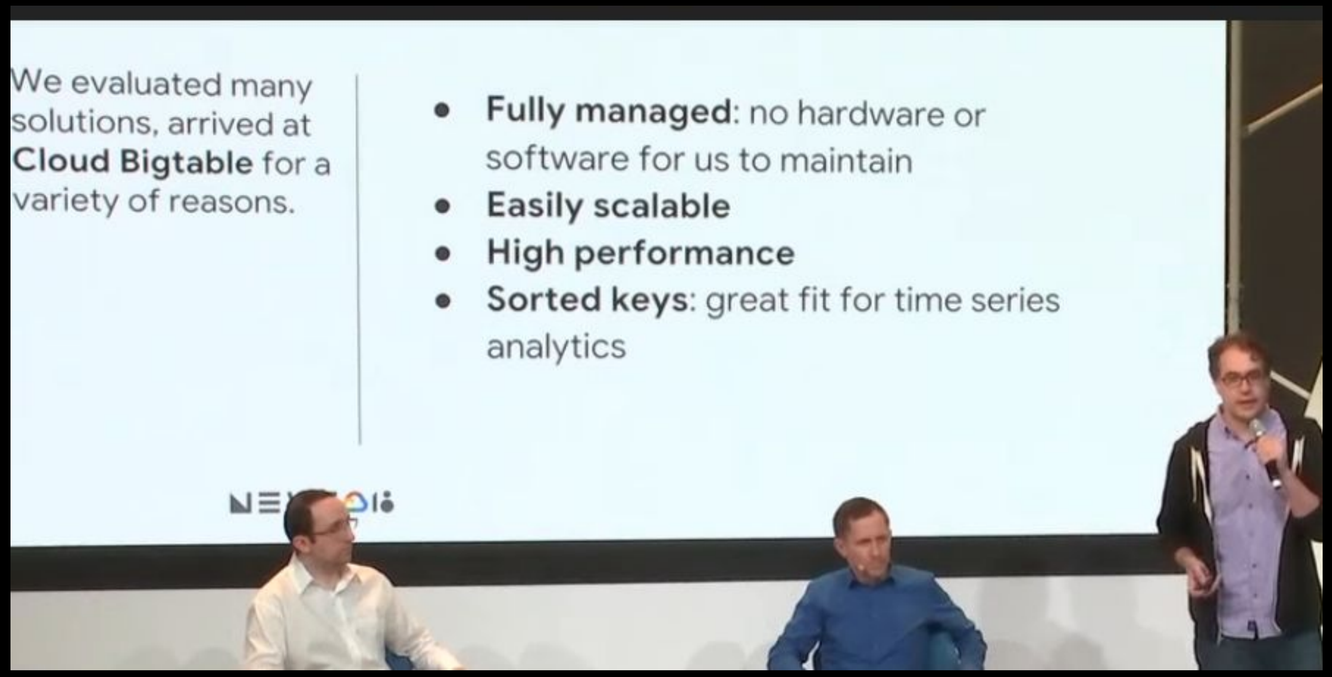


TESTING/TROUBLESHOOTING PERFORMANCE

- Use a production instance.
- Use at least 300 GB of data.
- Stay below the recommended storage utilization per node.
- Before you test, run a heavy pre-test for several minutes.
- Run your test for at least 10 minutes.

- Look at the Key Visualizer scans for your table.
- Try commenting out the code that performs Cloud Bigtable reads and writes.
- Ensure that you're creating as few clients as possible.
- Make sure you're reading and writing many different rows in your table.
- Verify that you see approximately the same performance for reads and writes.

REAL WORLD EXAMPLE



- 100 000s of active advertisers
- 100 000s of impressions / second
- ~20 TB / day of raw impression data
- ~8 TB / year of aggregated data
- Total dataset size is ~50 TB

- Analytics provided via advertiser-facing UI, API, and bulk-data exports
- Most queries need to be FAST
  - **SLA: p99 @ 4.5s**
  - **SLO: p99 @ 1s, p9999 @ 5s**
  - **Typical: p99 @ 300ms**
- High query volume: 5 000+ QPS, 1 GB/s data processed
- **High availability: uptime SLO of 99.97%**