Databend

The Modern Data Warehouse

https://github.com/datafuselabs/databend

BohuTANG(张雁飞)

Co-founder at Datafuse Labs

https://bohutang.me/

Agenda

- Traditional Data Warehouse
- Modern Data Warehouse
- Databend Design

Database and Data Warehouse

High Concurrency
Fast transaction
Low latency
Row oriented

Resource isolation?
Duplicate data?
Row+Column oriented

Complex query
Query speed
Column oriented

OLTP | HTAP | OLAP

MySQL etc.

TiDB etc.

ClickHouse etc.

Ideal Requirements

- No hardware to install, configure, or manage 

- No software to install, configure, or manage 

- No management, upgrades, and tuning 

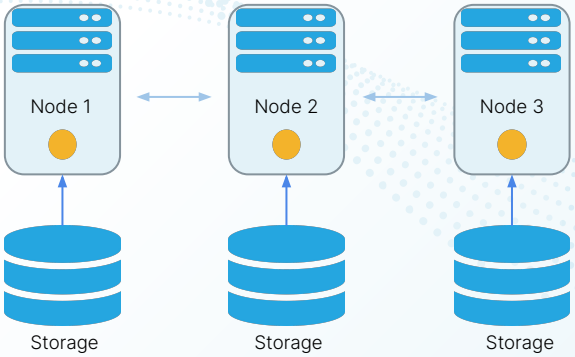
- Instant up- or down-scaling, elasticity 

- Pay only for used storage and compute resources 

Modern Data Warehouse

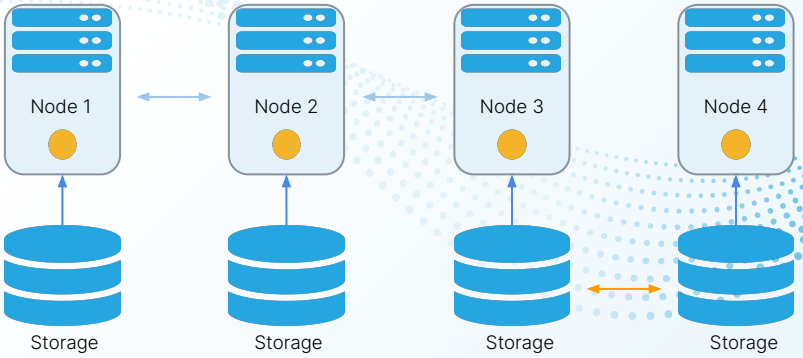
Traditional Data Warehouse

- Shared-nothing
- Compute(CPU/Mem), and storage aggregated
- Resource allocation coarse-grained
- Weak(not) elasticity



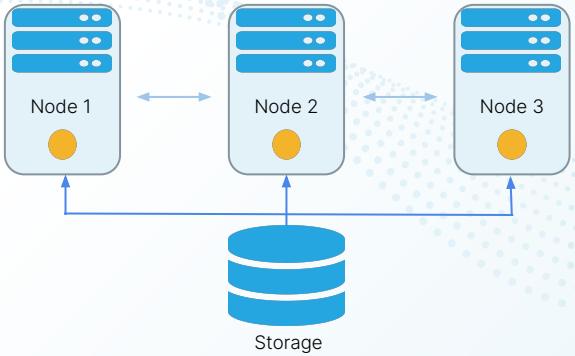
Traditional Data Warehouse

- Shared-nothing
- Compute(CPU/Mem), and storage aggregated
- Resource allocation coarse-grained
- Weak(not) elasticity



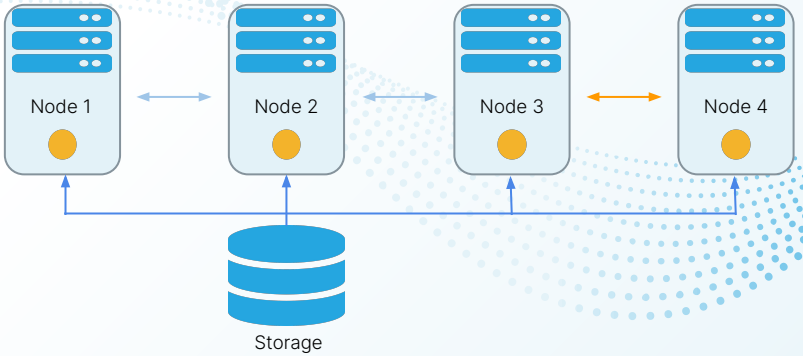
Modern Data Warehouse

- Shared-storage (elasticity)
- Compute and storage disaggregated (elasticity)
- Resource allocation fine-grained (elasticity)
- Instant up- or down-scaling (elasticity)



Modern Data Warehouse

- Shared-storage (elasticity)
- Compute and storage disaggregated (elasticity)
- Resource allocation fine-grained (elasticity)
- Instant up- or down-scaling (elasticity)



“

Hey, I want to run a query, run it as fast as you can, I only pay for the resources I use.

Cost = Resource * Time

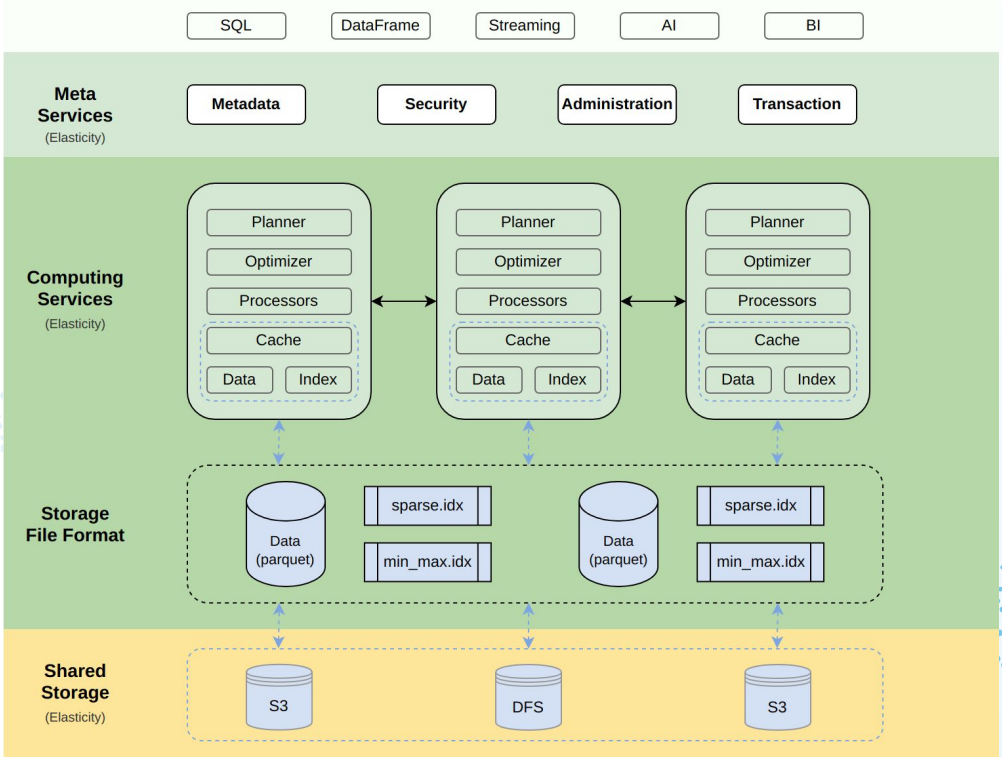
“

Hmm, Databend want to try elastic , elastic, elastic...

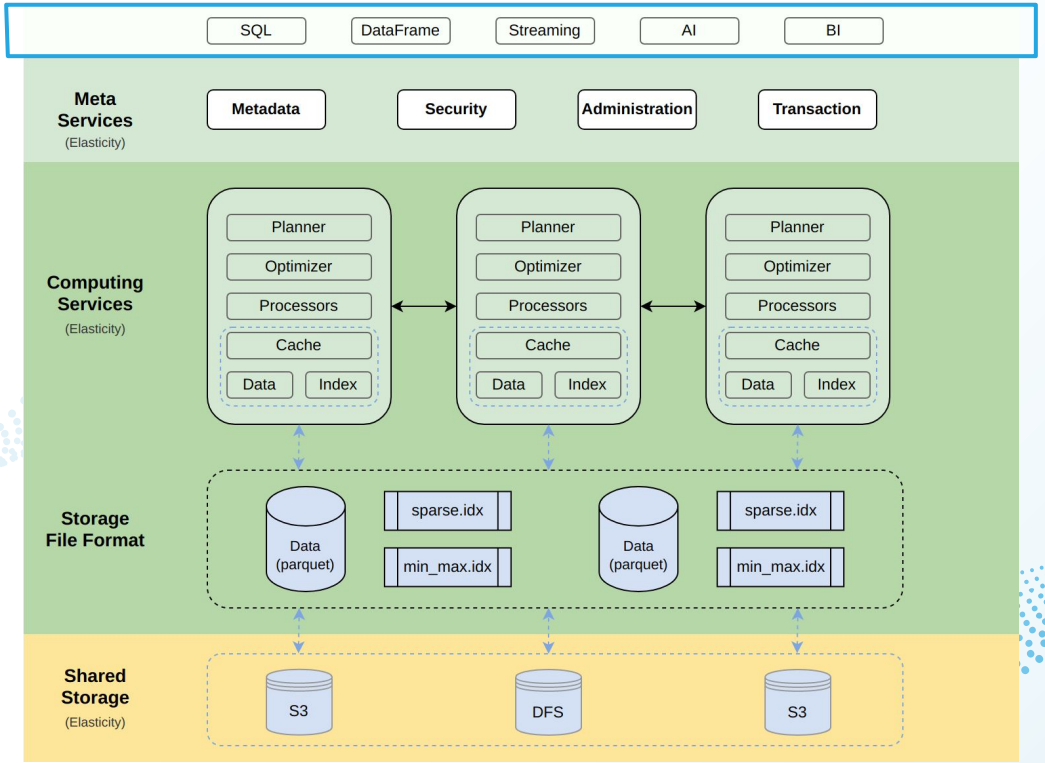
Databend

The modern(elastic) data warehouse

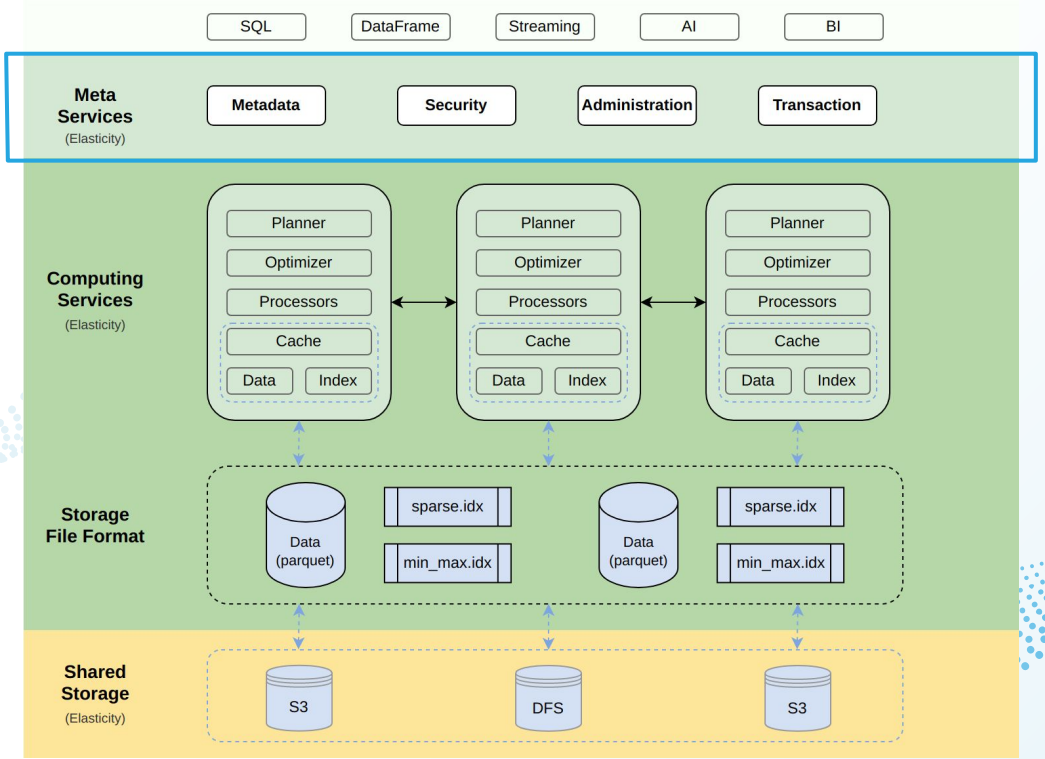
Databend Architecture



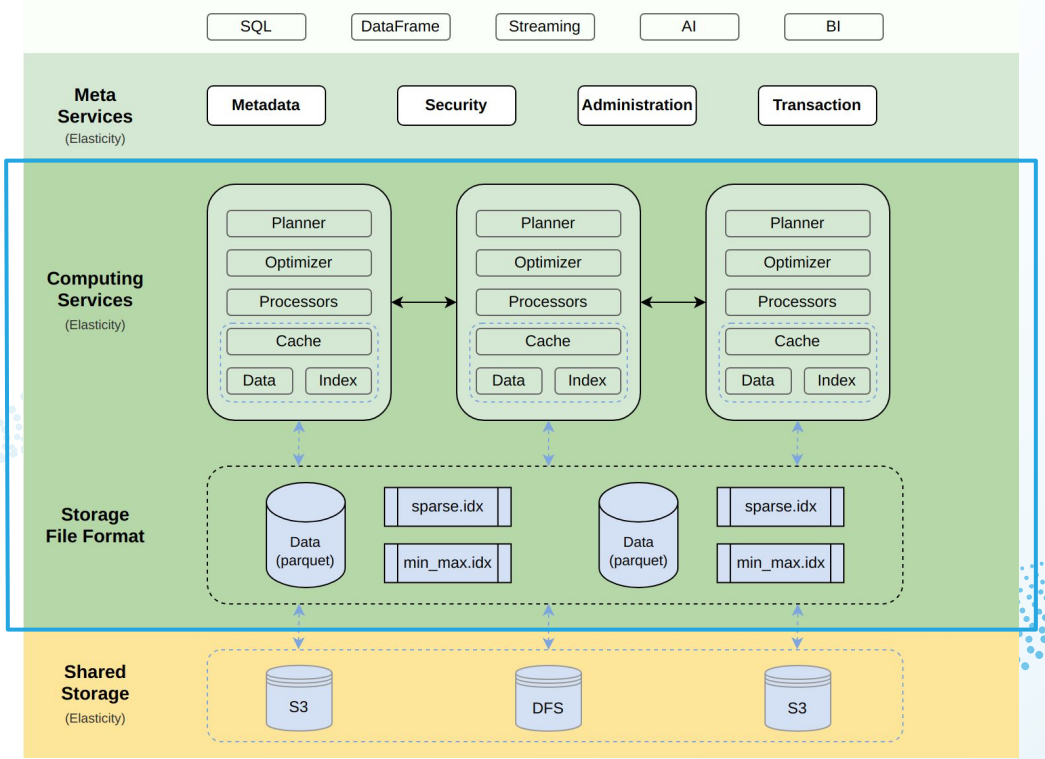
Databend Architecture



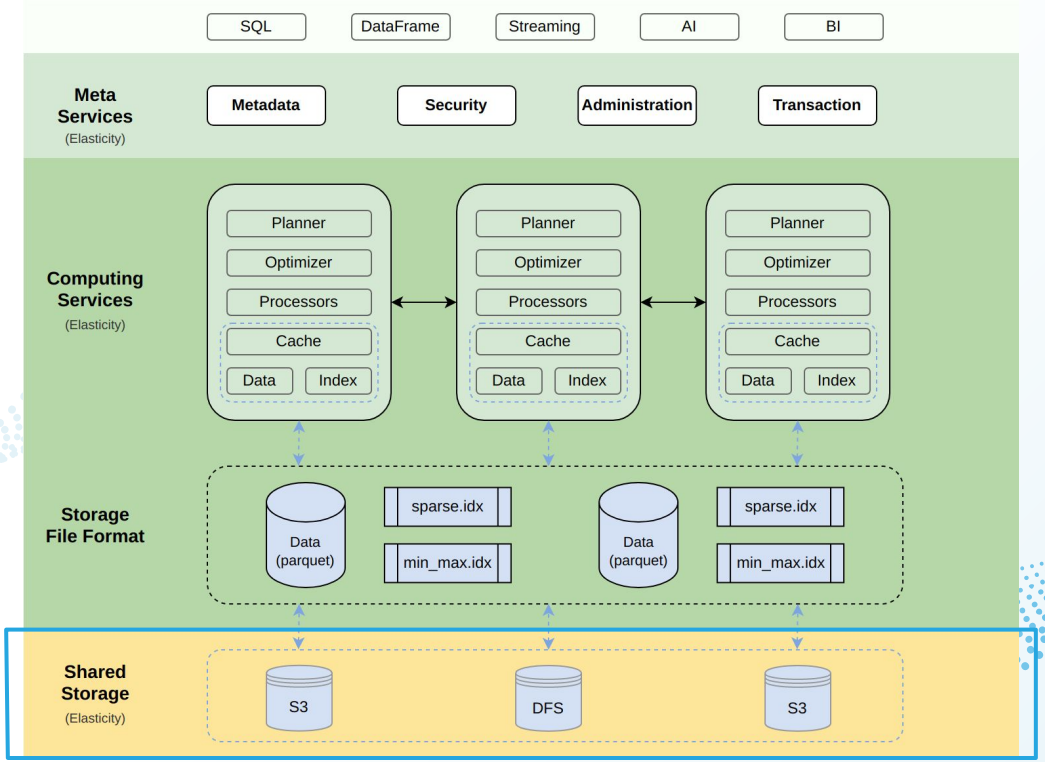
Databend Architecture



Databend Architecture



Databend Architecture



Meta Service

- Transactional Key-Value store
- User, Auth control information
- Database/Table/Index schema
- Cluster namespace registry

Compute: Query Planning

- SQL parser
- SQL planner
- Planner optimizer
- Distributed and Reshuffle data

explain **SELECT** avg(age) **FROM** class **WHERE** age > 13 **GROUP BY** city

```
┌─explain──────────────────────────────────────────────────────────────────────────────────────────────────────────────────┐
│ Projection: avg(age):Float64                                                                                                │
│   AggregatorFinal: groupBy=[[city]], aggr=[[avg(age)]]                                                                      │
│     AggregatorPartial: groupBy=[[city]], aggr=[[avg(age)]]                                                                  │
│       Filter: (age > 13)                                                                                                    │
│         ReadDataSource: scan partitions: [8], scan schema: [age:Int32, city:String], statistics: [read_rows: 20, read_bytes: 680] │
└──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘
```

Compute: Processor Pipeline

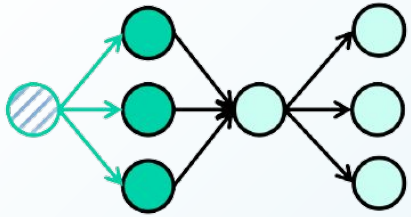
- Build from planner
- Directed Acyclic Graph (DAG) models
- Node is processor, Edge is channel
- Streaming connector: InPort and OutPort

explain pipeline **SELECT** avg(age) **FROM** class **WHERE** age > 13 **GROUP BY** city

```
┌─explain──────────────────────────────────────────────────────────────────────────────┐
│ ProjectionTransform × 8 processors                                                   │
│   Mixed (GroupByFinalTransform × 1 processor) to (ProjectionTransform × 8 processors) │
│     GroupByFinalTransform × 1 processor                                              │
│       Merge (GroupByPartialTransform × 8 processors) to (GroupByFinalTransform × 1)  │
│         GroupByPartialTransform × 8 processors                                       │
│           ExpressionTransform × 8 processors                                         │
│             SourceTransform × 8 processors                                           │
└──────────────────────────────────────────────────────────────────────────────────────┘
```

Compute: Pipeline Execution

- Pull based execution
- Vectorized execution(data)
- Parallel execution (CPUs)
- Distributed execution (cluster)
- Work-stealing scheduler



Compute: Cluster Execution

- Arrow Flight RPC
- **NOT** require deserialization

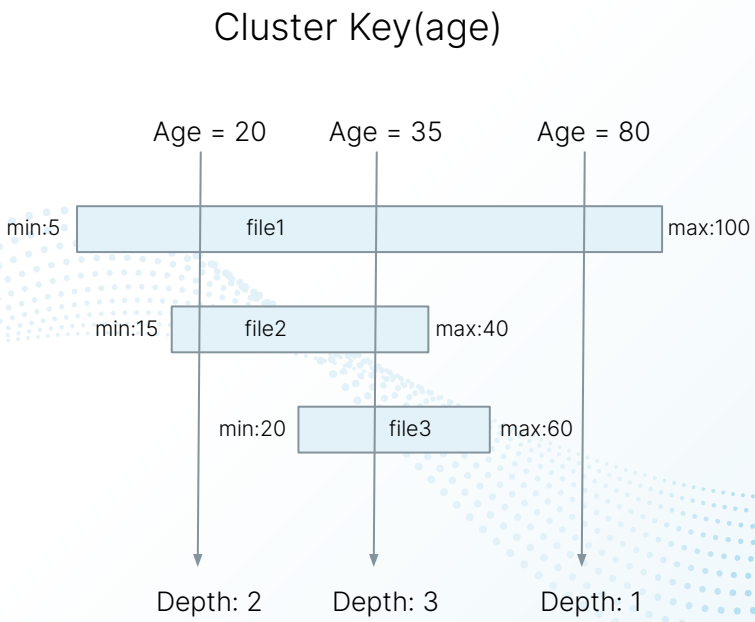
Storage Layer

- Column Storage
- Parquet format
- Index: MinMax, Sparse Index etc.

  Cloud Storage NOT design for low-latency and high-throughput
- Multi-Cloud, Single View of Data

  Cloud Storage (AWS S3, Azure Blob, Azure Data Lake, Google Cloud Storage)

Automatic Clustering



Databend Plan and Future

- Still in working(2021)
- Alpha version will be released soon
- Speed and Elastic
- On-Premise and On-Cloud both
- Serverless

Thanks!

- https://databend.rs

JOIN US

- hr@datafuselabs.com

  Databend kernel engineer

  Databend cloud engineer

Credits

- Presentation template by SlidesCarnival