Amping up Real-Time Query on Data Lakes with ClickHouse® and Project Antalya Swarms

Alexander Zaitsev - Altinity CTO
Robert Hodges - Altinity CEO

21 May 2025

Run Open Source ClickHouse® Better

Altinity.Cloud Enterprise Support

ClickHouse is a famous real-time analytic database

Understands SQL

Runs on bare metal to cloud

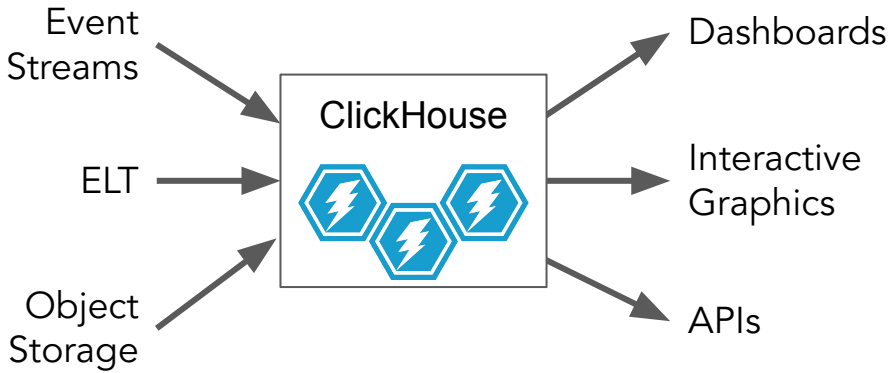
Shared nothing architecture

Stores data in columns

Parallel and vectorized execution

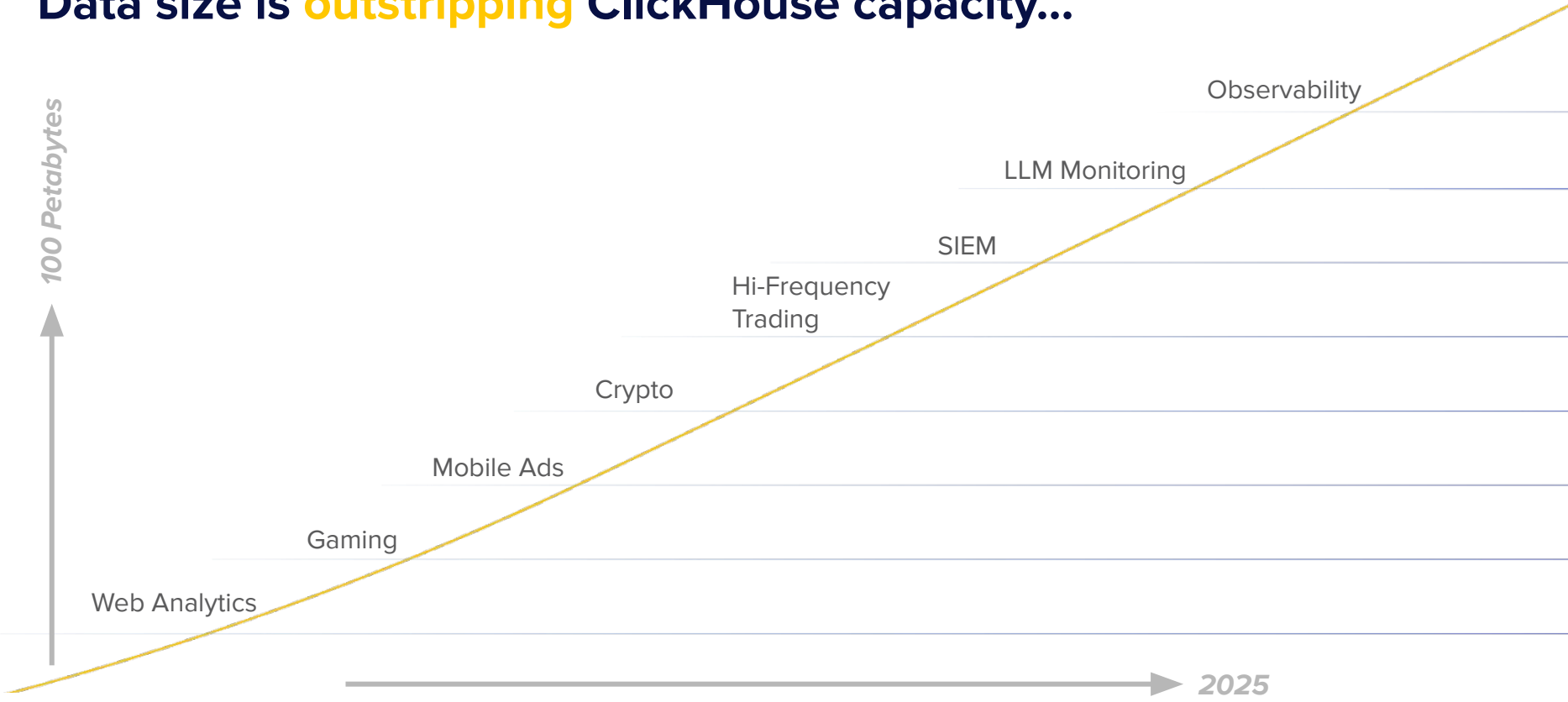
Scales to many petabytes

Is Open source (Apache 2.0)



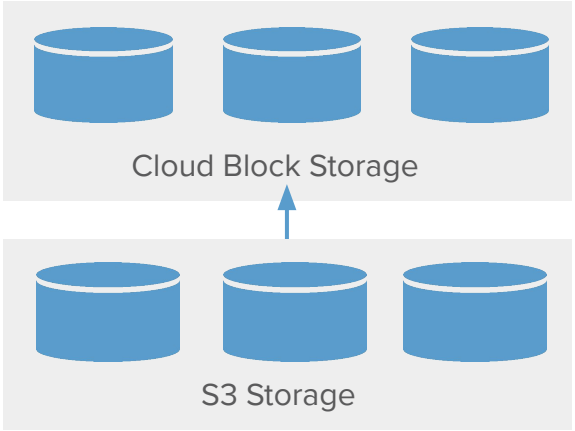
It's a popular engine for real-time analytics

Data size is outstripping ClickHouse capacity...

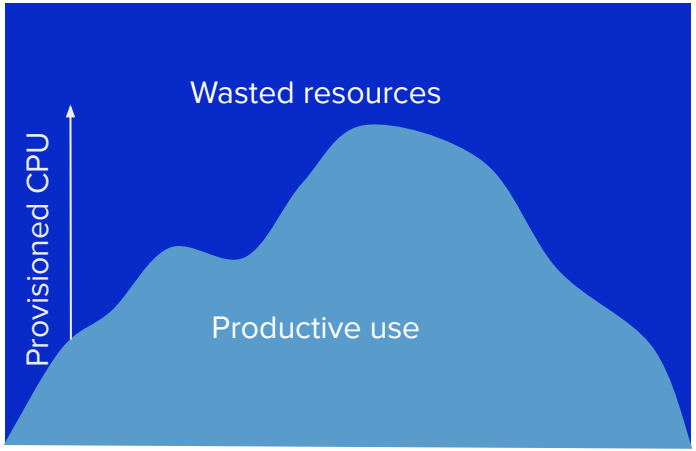


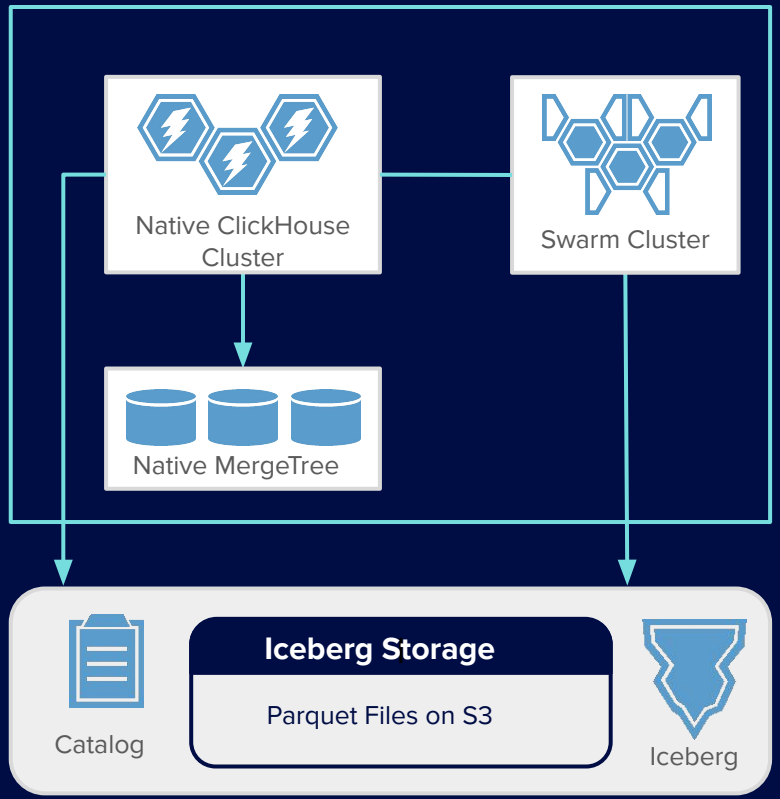
...Leading to pressure on storage and compute cost

Block storage with replication is 10x more expensive



Overprovisioning wastes compute

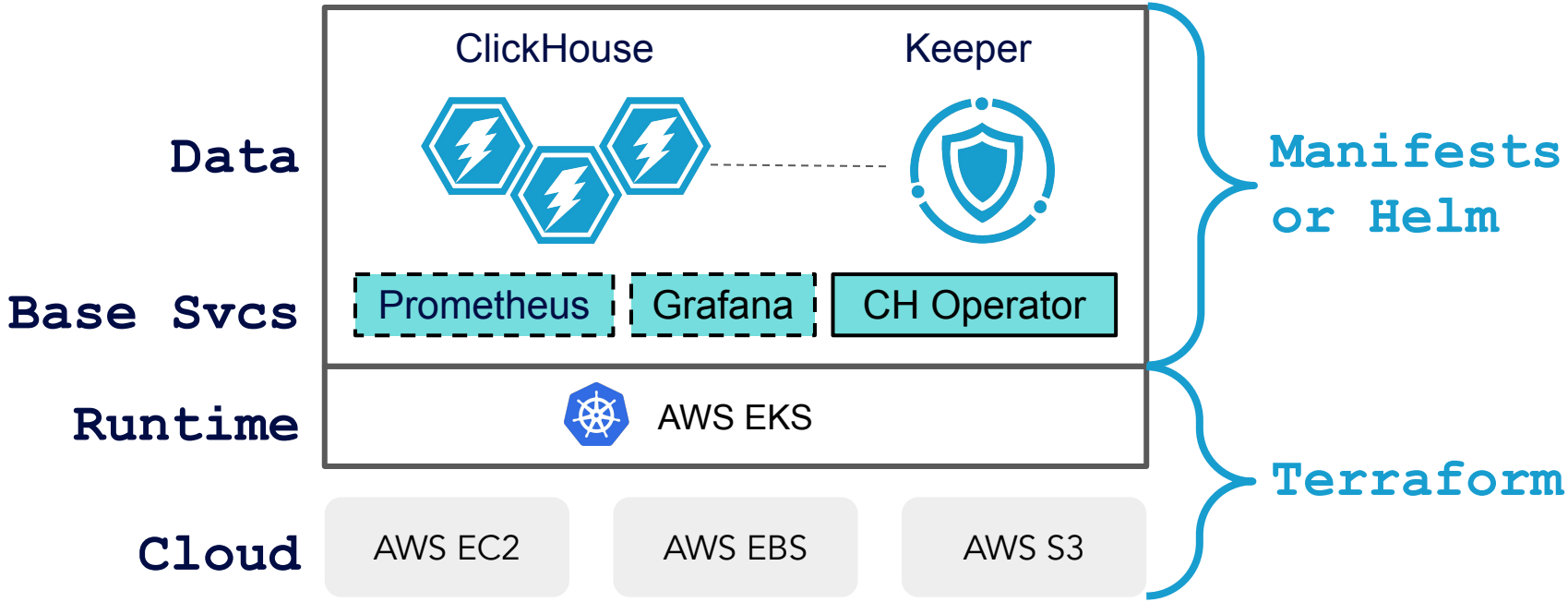




Project Antalya makes ClickHouse run fast and cheap on shared data

- Extends native ClickHouse capabilities
- Adds Iceberg for shared storage
- Adds swarm clusters for scalable compute
- 100% open source

Typical SaaS deployment – ClickHouse on AWS EKS



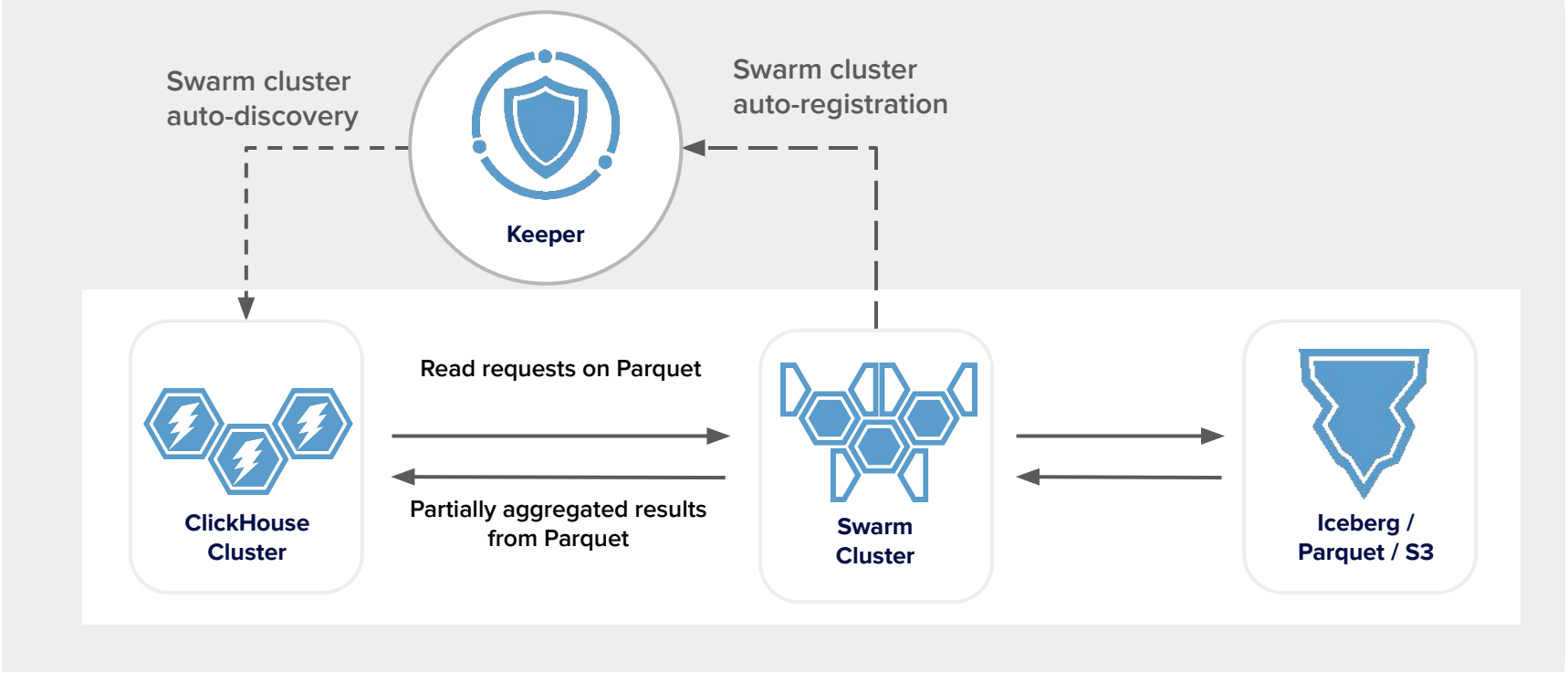
Let’s make a Kubernetes cluster using Terraform (EKS example)

1. Clone the antalya-examples repo.
   ```
   git clone https://github.com/Altinity/antalya-examples
   ```

2. Bring up Kubernetes using Terraform
   ```
   cd antalya-examples/kubernetes/terraform
   terraform init
   terraform apply
   ```

3. Update kubeconfig and start having fun.
   ```
   aws eks update-kubeconfig --name my-eks-cluster
   ```

Swarm clusters handle scans on object storage files



Bring up Project Antalya with a swarm cluster using manifests

1. Cd to manifests director from terraform directory.
   ```
   cd ../manifests
   ```

2. Load manifests for servers.
   ```
   kubectl apply -f gp3-encrypted-fast-storage-class.yaml
   kubectl apply -f keeper.yaml
   kubectl apply -f swarm.yaml
   kubectl apply -f vector.yaml
   ```

3. Test out your new cluster. This shows the swarm clusters are reachable.
   ```
   kubectl exec -it chi-vector-example-0-0-0 -- clickhouse-client \
   "SELECT hostName(), version() FROM clusterAllReplicas('swarm',
   system.one)"
   ```

Details: Swarm servers are stateless ClickHouse servers

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "swarm"
spec:
  configuration:
    clusters:
      - name: "example"
        layout:
          shardsCount: 4
          replicas:
          - templates:
              podTemplate: replica
              volumeClaimTemplate: storage
    zookeeper:
        nodes:
        - host: keeper-keeper
          port: 2181
```

Change shardsCount to scale swarm

Creates a single replica per shard

[Zoo]Keeper for auto-discovery

More details: Configure auto-discovery using the <discovery> tag

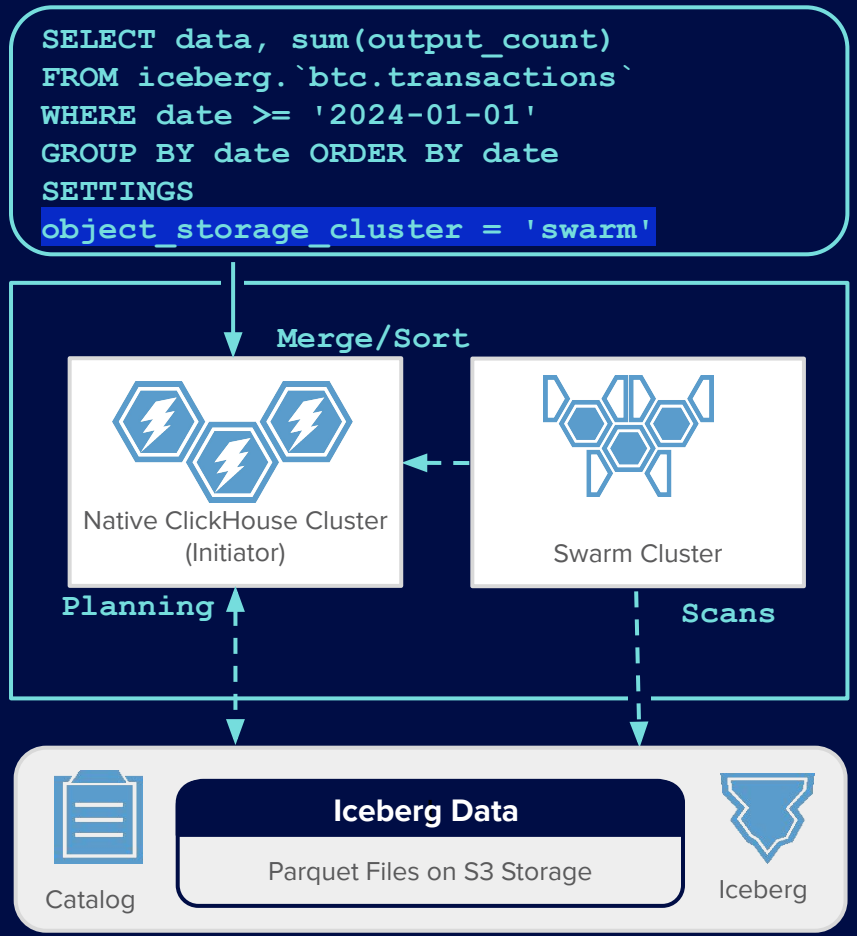
```
config.d/remote_servers.xml: |
<clickhouse>
  <allow_experimental_cluster_discovery>1</allow_experimental_cluster_discovery>
  <remote_servers>
    <!-- Swarm cluster built using remote discovery. -->
    <swarm>
      <discovery>
        <path>/clickhouse/discovery/swarm</path>
        <secret>secret_key</secret>
        <!-- Swarm servers are false, initiators calling swarm are true. -->
        <observer>false</observer>
      </discovery>
    </swarm>
  </remote_servers>
</clickhouse>
```

Provides auto-discovery path in Keeper

Initiators calling the swarm set this to **true**

Project Antalya query model

- Initiator plans the query
- Swarm Cluster nodes scan shared files
- Initiator merges and sorts responses



Queries directly on Hive tables on S3 using swarm cluster

```
SELECT date, sum(output_count)
FROM s3('s3://aws-public-blockchain/v1.0/btc/transactions/**.parquet',
  NOSIGN)
WHERE date >= '2025-01-01'
GROUP BY date ORDER BY date ASC
SETTINGS
  use_hive_partitioning = 1,
  object_storage_cluster = 'swarm',
  object_storage_max_nodes = 2;
```

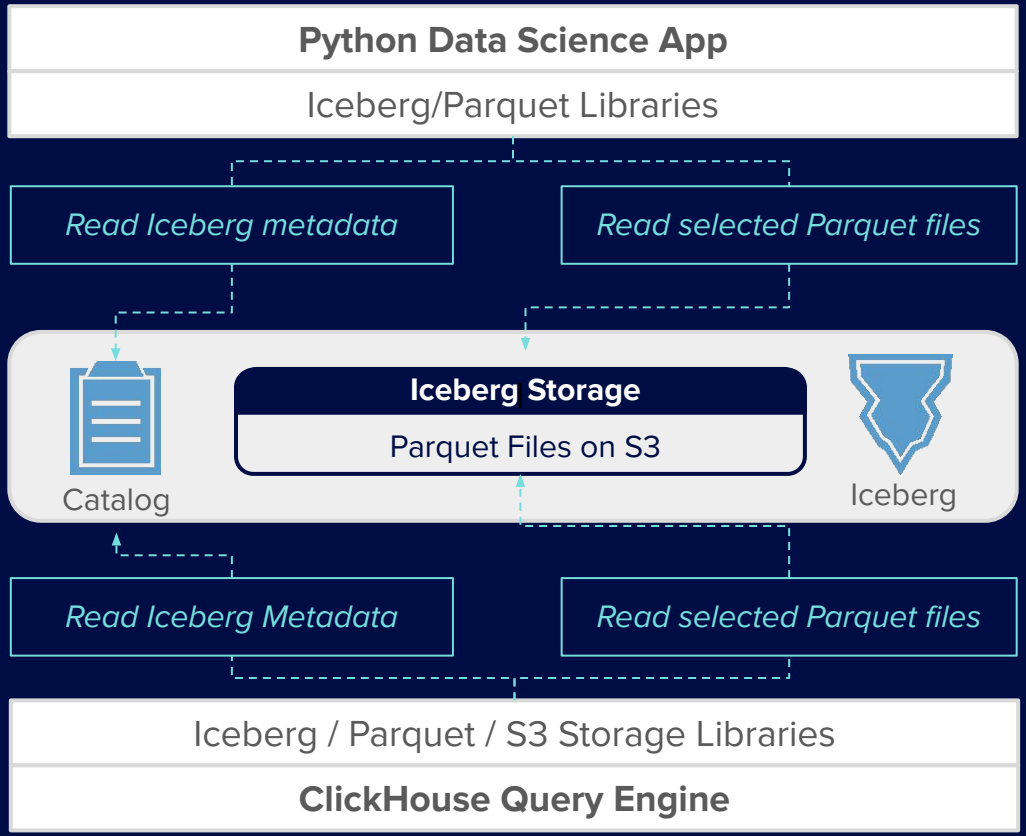
File paths use hive partitioning

Swarm cluster to use

Use just 2 nodes not all

A quick detour into Iceberg data lakes

- There's no database!
- Catalog serves Iceberg table metadata
- Table files are on S3



Connecting ClickHouse to a REST catalog

```
CREATE DATABASE ice ENGINE =
  Iceberg('https://iceberg-catalog.your-company.com')
SETTINGS
  catalog_type = 'rest',
  auth_header = 'Authorization: Bearer 0**********************************8',
  warehouse = 's3://some-bucket-path';

SHOW TABLES FROM ice;
```

```
    ┌─name──────────────────────────────┐
1.  │ aws-public-blockchain.btc         │
2.  │ aws-public-blockchain.btc_ps_by_date │
    └───────────────────────────────────┘
```

Catalog server URL

Cache Parquet file metadata

Path to Iceberg metadata and data files

Selecting data from Iceberg tables is way easier

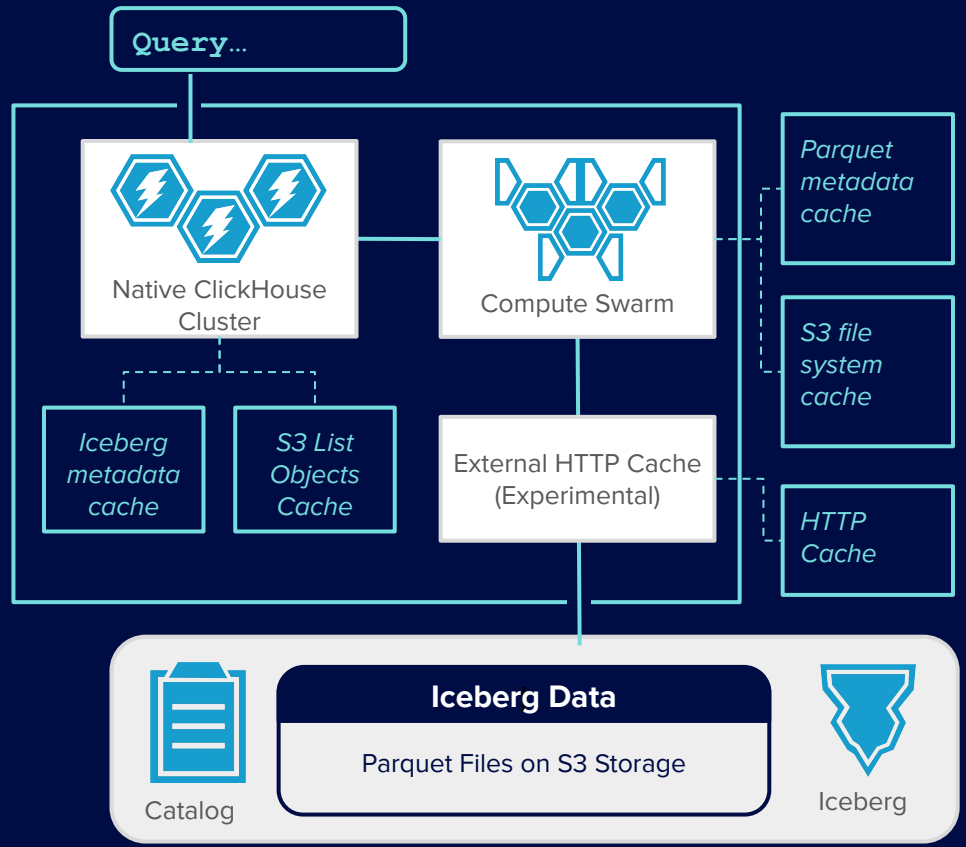
```
SELECT
    date,
    sumDistinct(block_number) AS blocks,
    sum(fee) AS fees,
    fees / blocks AS fee_per_block
FROM ice.`aws-public-blockchain.btc`
WHERE date >= '2025-01-01'
GROUP BY date
ORDER BY date ASC
SETTINGS object_storage_cluster = 'swarm'
```

Backticks required on iceberg table names

Catalog server URL

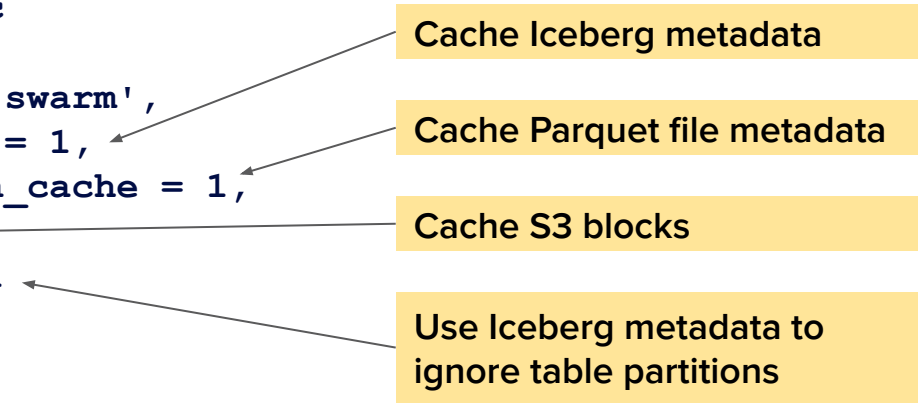
Caches omit needless I/O

- Skip parsing Iceberg metadata
- Skip calls to list files in object storage
- Skip Parquet files entirely
- Skip reading file metadata
- Skip fetching contents from S3



Key settings for query performance on Iceberg tables

```
SELECT
    min(date),
    max(date)
FROM ice.`aws-public-blockchain.btc`
WHERE date >= '2025-02-01'
SETTINGS object_storage_cluster = 'swarm',
  use_iceberg_metadata_files_cache = 1,
  input_format_parquet_use_metadata_cache = 1,
  enable_filesystem_cache = 1,
  use_iceberg_partition_pruning = 1
```



Key settings for performance on Hive tables and plain S3 files

```
SELECT
    date,
    count()
FROM s3('s3://aws-public-blockchain/v1.0/btc/transactions/*/*.parquet',
NOSIGN)
WHERE (date >= '2025-01-01') AND (date <= '2025-01-31')
GROUP BY date
ORDER BY date ASC
SETTINGS
  use_hive_partitioning = 1,
  use_object_storage_list_objects_cache = 1
```

File paths use Hive format

Cache ListObjectsV2 calls to list S3 files (way faster than it sounds!)

Enabling filesystem caches for S3

```
config.d/filesystem_cache.xml: |
  <clickhouse>
    <filesystem_caches>
      <s3_parquet_cache>
        <path>/var/lib/clickhouse/s3_parquet_cache</path>
        <max_size>50Gi</max_size>
      </s3_parquet_cache>
    </filesystem_caches>
  </clickhouse>
```

Path and size on on disk

Name of the cache

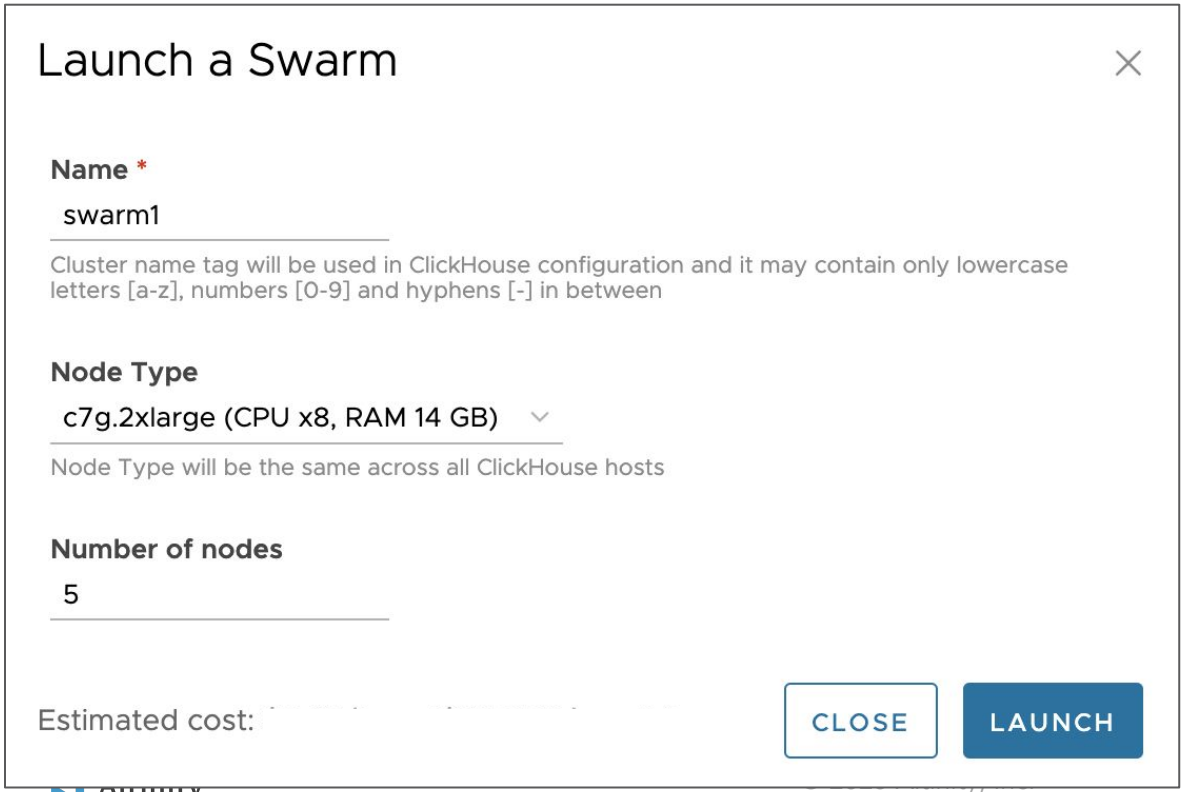
Query settings are complex, so put them in a profile!

```
$ cat /etc/clickhouse-server/users.d/chop-generated-profiles.xml
<clickhouse>
    <profiles>
        <cache_enabled>
            <enable_filesystem_cache>1</enable_filesystem_cache>
            <filesystem_cache_name>s3_parquet_cache</filesystem_cache_name>
            <remote_filesystem_read_prefetch>0</remote_filesystem_read_prefetch>
        </cache_enabled>
        <default>
            <enable_filesystem_cache>0</enable_filesystem_cache>
            <filesystem_cache_name>s3_parquet_cache</filesystem_cache_name>
            <remote_filesystem_read_prefetch>0</remote_filesystem_read_prefetch>
        </default>
    </profiles>
</clickhouse>
```

Running Antalya Swarms on Altinity.Cloud (UI preview)

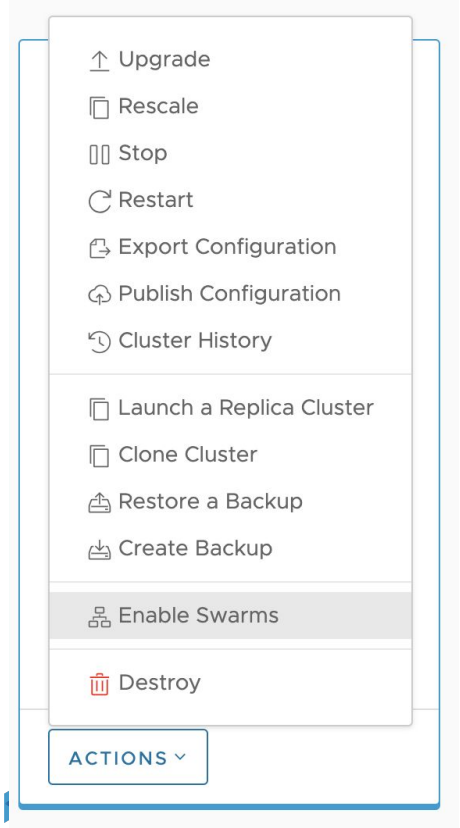
1. Starting a swarm.
2. Connecting your cluster to a swarm.
3. Connecting to Iceberg tables
4. Running queries

Starting a Swarm in Antalya.Cloud



- Automatically starts registry
- Uses latest Antalya build
- Configures for the best performance
- Configures swarm self-registration

Enabling Swarm access to an existing ClickHouse

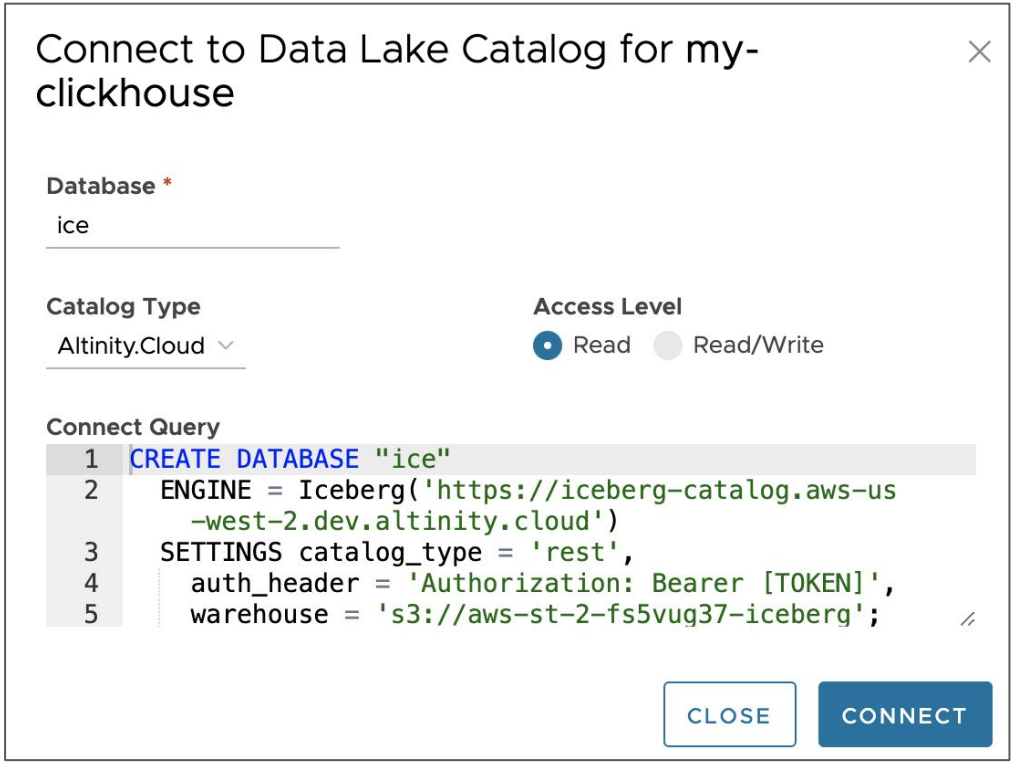


Requires upstream 25.3+ or Antalya build for registry to work

```
select name, host_name from system.clusters
```

```
    ┌─name──────────┬─host_name──────────────────────────────────────┐
1.  │ all-clusters  │ chi-my-clickhouse-my-clickhouse-0-0            │
2.  │ all-replicated│ chi-my-clickhouse-my-clickhouse-0-0            │
3.  │ all-sharded   │ chi-my-clickhouse-my-clickhouse-0-0            │
4.  │ default       │ localhouse                                     │
5.  │ my-clickhouse │ chi-my-clickhouse-my-clickhouse-0-0            │
6.  │ swarm1        │ chi-swarm1-swarm1-0-0-0.chi-swarm1-swarm1-0-0  │
7.  │ swarm1        │ chi-swarm1-swarm1-1-0-0.chi-swarm1-swarm1-1-0  │
8.  │ swarm1        │ chi-swarm1-swarm1-2-0-0.chi-swarm1-swarm1-2-0  │
9.  │ swarm1        │ chi-swarm1-swarm1-3-0-0.chi-swarm1-swarm1-3-0  │
10. │ swarm1        │ chi-swarm1-swarm1-4-0-0.chi-swarm1-swarm1-4-0  │
    └───────────────┴────────────────────────────────────────────────┘
```

Connecting your cluster to a data lake catalog




```
show tables from ice

┌─name──────────────────────────────────┐
1. │ aws-public-blockchain.btc             │
2. │ aws-public-blockchain.btc_ps_by_date  │
3. │ flowers.iris                          │
4. │ nyc.taxis                             │
   └───────────────────────────────────────┘
```

Running Queries

```
SELECT date, sum(output_count)
FROM ice."aws-public-blockchain.btc_ps_by_date"
WHERE date >= '2025-01-01'
GROUP BY date ORDER BY date
SETTINGS object_storage_cluster='swarm1'
```

(query time: 0.541s, read rows: 51578686, read bytes: 3267070692)

```
    ┌─date───────┬─sum(output_count)─┐
 1. │ 2025-01-01 │            849704 │
 2. │ 2025-01-02 │           1084152 │
 3. │ 2025-01-03 │           1139570 │
 4. │ 2025-01-04 │           1124689 │
```

Using Spot Instances

The good:

- Cost ***up to*** 90% less than on-demand. In reality one can expect 50%
- Quick scale up/down, pay for seconds used (as opposed to full hours)

The bad:

- May be interrupted/shut-down
- May be not available when needed

Starting Swarms Fast

ClickHouseOperatorConfiguration – applies to all CHI:

```
spec:
  reconcile:
    runtime:
       reconcileShardsThreadsNumber: 100
       reconcileShardsMaxConcurrencyPercent: 100
```

ClickHouseInstallation (operator version >= 0.25.0) – applies to specific CHI:

```
spec:
  reconciling:
    runtime:
      reconcileShardsMaxConcurrencyPercent: 100
```

Performance Introspection

```
SELECT arrayJoin(
         mapFilter((k, v) -> (k LIKE 'S3%'
OR k LIKE 'Iceberg%' OR k LIKE 'Parquet%'),
           sumMap(ProfileEvents))
       ) AS s3_events
  FROM system.query_log
 WHERE type = 'QueryFinish'
   AND event_date = today()
```

```
    ┌─s3_events──────────────────────────────────────┐
 1. │ ('IcebergMetadataFilesCacheHits',78)           │
 2. │ ('ParquetFetchWaitTimeMicroseconds',65016013)  │
 3. │ ('ParquetMetaDataCacheHits',2827)              │
 4. │ ('ParquetMetaDataCacheMisses',241)             │
 5. │ ('S3Clients',45)                               │
 6. │ ('S3GetObject',3038)                           │
 7. │ ('S3HeadObject',3844)                          │
 8. │ ('S3ListObjects',60)                           │
 9. │ ('S3ReadMicroseconds',211318435)               │
10. │ ('S3ReadRequestsCount',7022)                   │
11. │ ('S3ReadRequestsErrors',40)                    │
12. │ ('S3WriteMicroseconds',517199)                 │
13. │ ('S3WriteRequestsCount',19)                    │
```

```
select * from system.events where name ilike '%iceberg%';
select * from system.metrics where name ilike '%iceberg%';
select * from system.asynchronous_metrics where metric ilike '%iceberg%';
```

Explore Catalog Internals

```
select _path, any(_size)
from ice."aws-public-blockchain.btc_ps_by_date"
group by 1 with totals
```

- Lists all files and sizes
- Does not use catalog metadata yet

Querying catalog contents directly (requires IAM access)

```
select date, count()
from ice."aws-public-blockchain.btc_ps_by_date"
where date between '2025-01-01' and '2025-01-31'
group by 1
```

```
select date, count()
from
s3('s3://aws-st-2-fs5vug37-iceberg/aws-public-blockchain/btc_ps_by_date
/data/date=*/*.parquet')
where date between '2025-01-01' and '2025-01-31'
group by 1
```

Matches namespace and table name

May be missing, hive-partitioning for convenience only

Upstream ClickHouse vs Project Antalya

	Upstream	Antalya
Swarm discovery	25.3+	yes
Catalog databases	yes	yes
Hosted catalogs	n/a	yes
Swarm queries	Partially, using table functions	yes
Parquet metadata cache	not supported	yes
Iceberg metadata cache	25.4+	yes
Pre-tuning for optimal performance	n/a	yes

Project Antalya Roadmap

1. ice – open source tools for loading and running Iceberg REST catalogs – 05/25
2. Support for AWS S3 tables - 06/25
3. Altinity.Cloud release with catalog and swarms support – 06/25
4. Performance report – 06/25
5. Writing to Iceberg tables – Q3/25
6. Tiered MergeTree+Iceberg tables – Q4/25

Project Antalya resources

- Check out Altinity antalya-examples repo for samples and documentation

  `git@github.com:Altinity/antalya-examples.git`

- Project Antalya code is in the Altinity ClickHouse repo (log issues there)

  `git@github.com:Altinity/ClickHouse.git`

- Read “Getting Started with Altinity’s Project Antalya” to find out more

  `https://altinity.com/blog/getting-started-with-altinitys-project-antalya`

- Join the Altinity Public Slack to find out more: `https://altinity.com/slack`

Summary

- Project Antalya is extending ClickHouse to use Iceberg as table storage
- Swarm clusters enable compute/storage separation on reads now
- Caches are vital to ensure performance
- Additional tricks include spot instances, fast operator reconciliation, and diagnostic queries
- Upcoming attractions:
  - Altinity.Cloud users can run Project Antalya clusters starting in June
  - Watch out for ice, a project to run and load Iceberg REST catalogs easily

Watch for our in-person meetup on ClickHouse and Real-Time Data Lakes

Where: Sentry offices in San Francisco
When: Tues 8 July 2025

Thank you!
Questions?

Contact us to learn more and join our community:

https://altinity.com

https://altinity.com/slack

https://github.com/Altinity/antalya-examples

ALTINITY