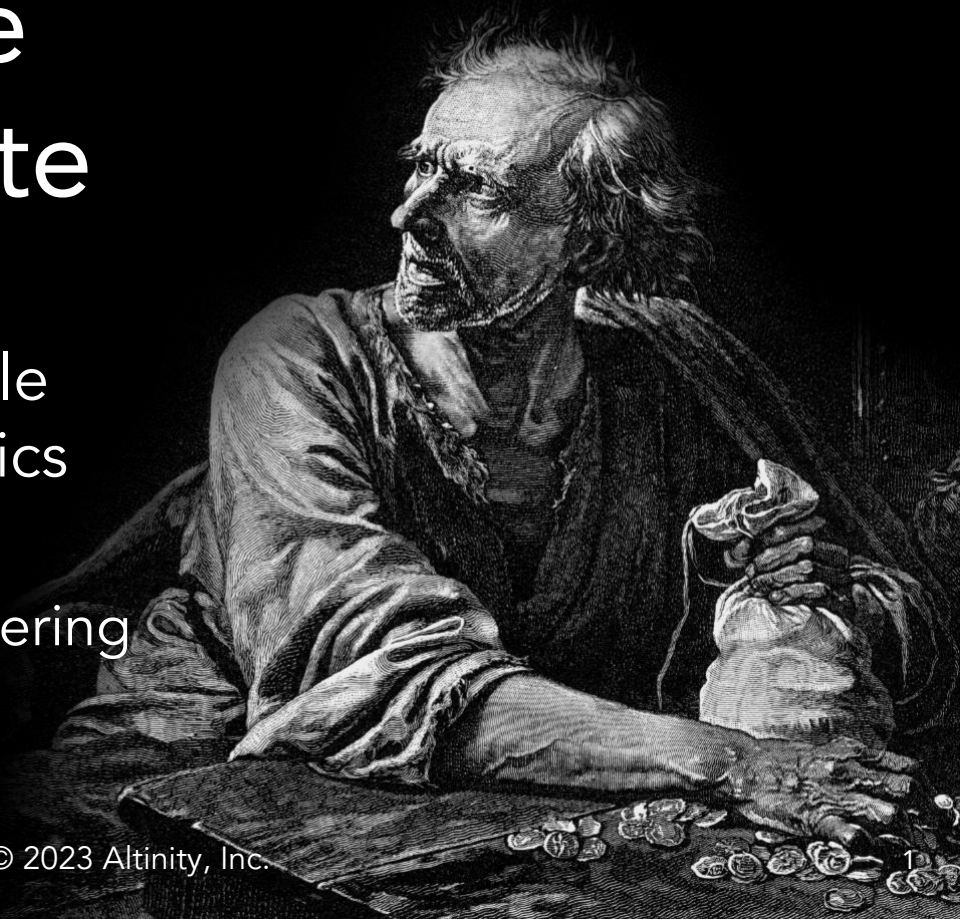


# Run ClickHouse Like a Cheapskate

6 Ways to Save Money While  
Delivering Real-Time Analytics

Robert Hodges & Altinity Engineering



# Let's make some introductions

## Robert Hodges

Database geek with 30+ years  
on DBMS systems. Day job:  
Altinity CEO

## Altinity Engineering

Database geeks with centuries  
of experience in DBMS and  
applications



ClickHouse support and services including [Altinity.Cloud](#)  
Authors of [Altinity Kubernetes Operator for ClickHouse](#)  
and other open source projects

# And ClickHouse, a real-time analytic database

Understands SQL

Runs on bare metal to cloud

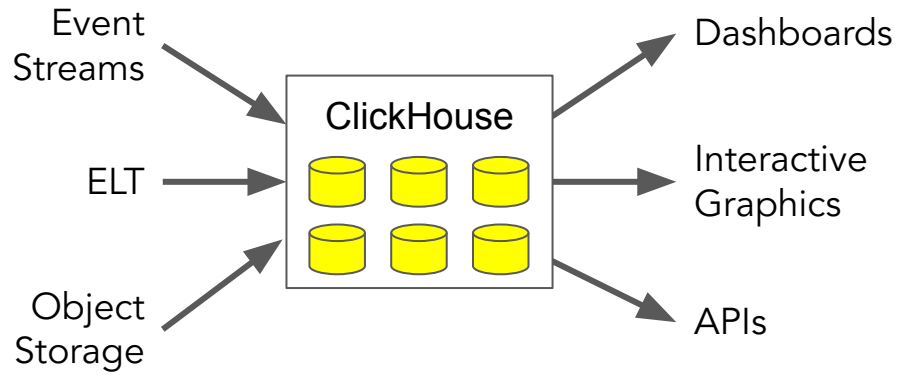
Shared nothing architecture

Stores data in columns

Parallel and vectorized execution

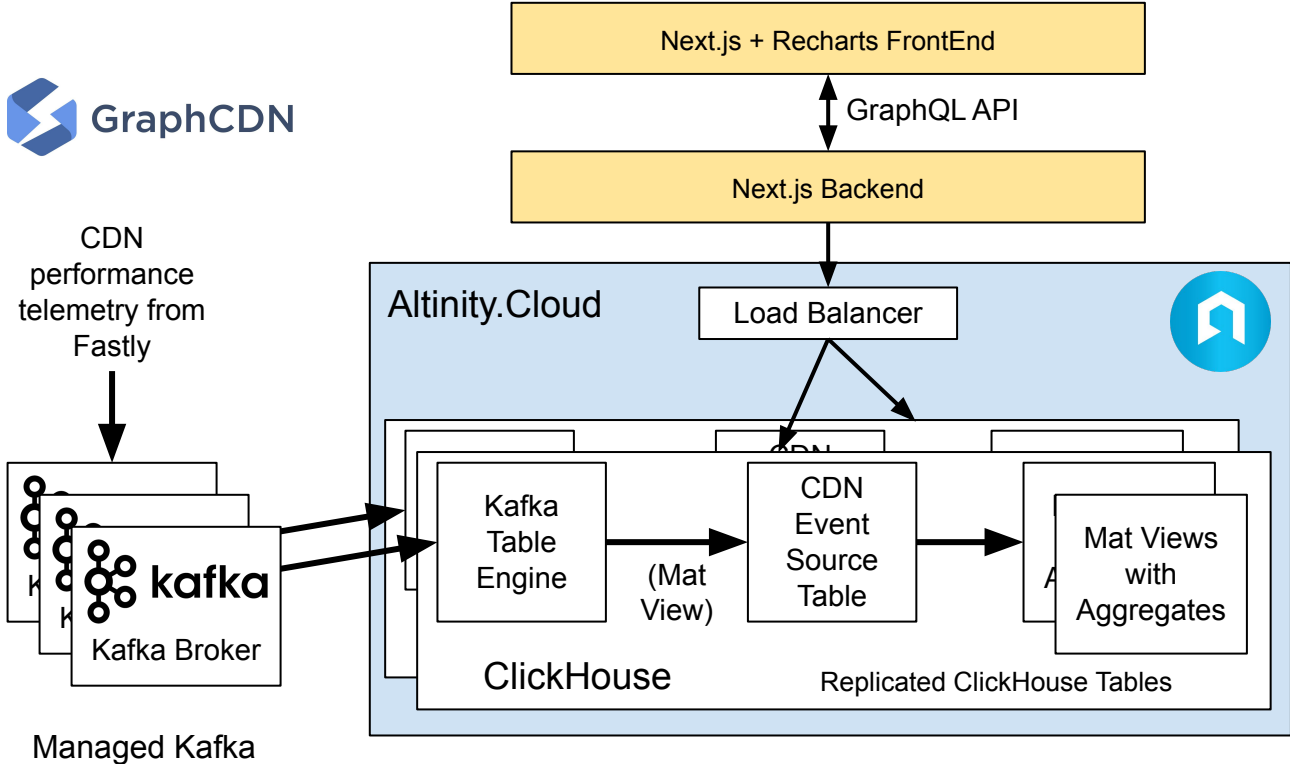
Scales to many petabytes

Is Open source (Apache 2.0)

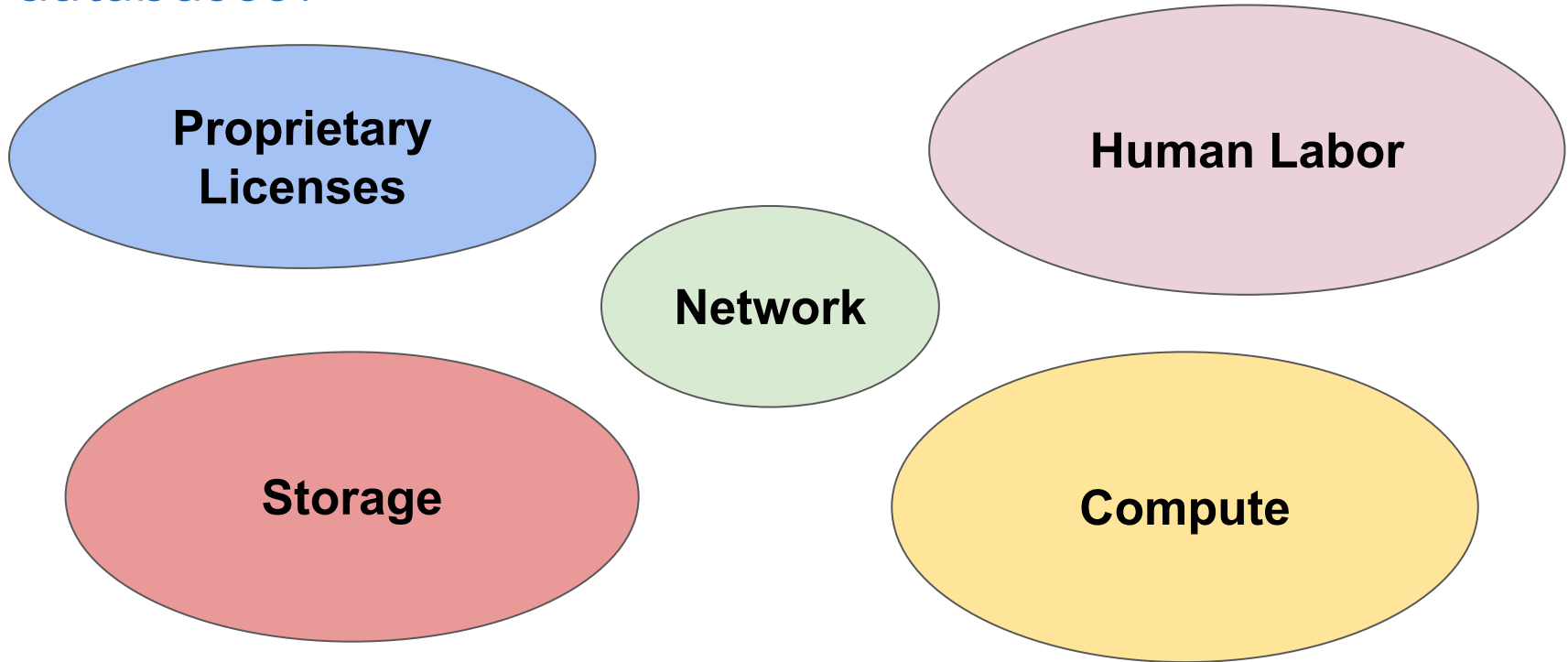


It's the core engine for  
low-latency analytics

# Practical example of ClickHouse in a real application



# What are the principle operating costs in analytic databases?

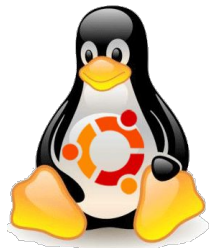


Develop on a  
laptop with 100%  
open source

# Three open source dev patterns that work on a laptop

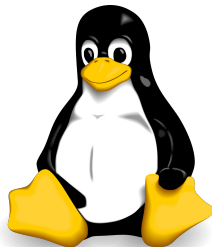
Install ClickHouse packages directly

```
sudo apt install -y  
clickhouse-server  
clickhouse-client
```



Run ClickHouse using docker

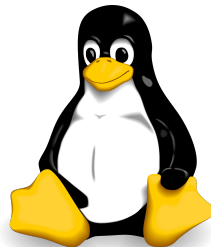
```
docker run -d --name  
clickhouse-server  
...
```



Mac OS

Run complete ClickHouse app with docker compose

```
docker compose up -d
```



Mac OS

# Let's build a ClickHouse "app" with Ubuntu + curl

```
cat > Dockerfile << END  
# Simple analytic client with curl installed.  
FROM ubuntu:22.04  
RUN apt-get update && apt-get install -y curl  
CMD ["sleep", "infinity"]  
END
```

```
docker build -t myclient:latest - < Dockerfile
```



# Create a compose file for ClickHouse and the app

```
cat > docker-compose.yml << END
version: '3'
services:
  clickhouse_server:
    image: altinity/clickhouse-server:22.8.15.25.altinitystable
    ports:
      - "8123:8123"
      - "9000:9000"
    volumes:
      - ./clickhouse_database:/var/lib/clickhouse
  ubuntu_client:
    image: myclient:latest
END
```

## ...And log into your “application”

```
$ docker compose up -d
```

```
...
```

```
$ docker ps
```

```
CONTAINER ID    ...    NAMES  
dff28a725b38    altinity/clickh...    cheapskate-clickhouse_server-1  
23a641654ac2    myclient:latest...    cheapskate-ubuntu_client-1
```

```
$ docker exec -it 23a6 bash
```

```
root@23a641654ac2:/# curl http://cheapskate-clickhouse_server-1:8123  
Ok.
```

# Use open source components to build your apps



## Event streaming

- [Apache Kafka](#)
- [Apache Pulsar](#)
- [Vectorized Redpanda](#)

## ETL

- [Apache Airflow](#)
- [Apache Nifi](#)
- [Rudderstack](#)

## Rendering/Display

- [Apache Superset](#)
- [Cube.js](#)
- [Grafana](#)

## Client Libraries

- C++ - [ClickHouse CPP](#)
- Golang - [ClickHouse Go](#)
- Java - [ClickHouse JDBC](#)
- Javascript/Node.js - [Apla](#)
- ODBC - [ODBC Driver for ClickHouse](#)
- Python - [ClickHouse Driver](#), [ClickHouse SQLAlchemy](#)

More client library links [HERE](#)

## Kubernetes

- [Altinity Operator for ClickHouse](#)

# How much can we save?

Develop on a laptop with 100% open source

Savings on  
Licensing: 100%



# Tune apps to limit resource usage

# Invest in schema design to reduce storage and I/O

```
CREATE TABLE IF NOT EXISTS readings_zstd (  
  sensor_id Int32 Codec(DoubleDelta, ZSTD(1)),  
  sensor_type UInt16 Codec(ZSTD(1)),  
  location LowCardinality(String) Codec(ZSTD(1)),  
  time DateTime Codec(DoubleDelta, ZSTD(1)),  
  date ALIAS toDate(time),  
  temperature Decimal(5,2) Codec(T64, ZSTD(10))  
)  
Engine = MergeTree  
PARTITION BY toYYYYMM(time)  
ORDER BY (location, sensor_id, time);
```

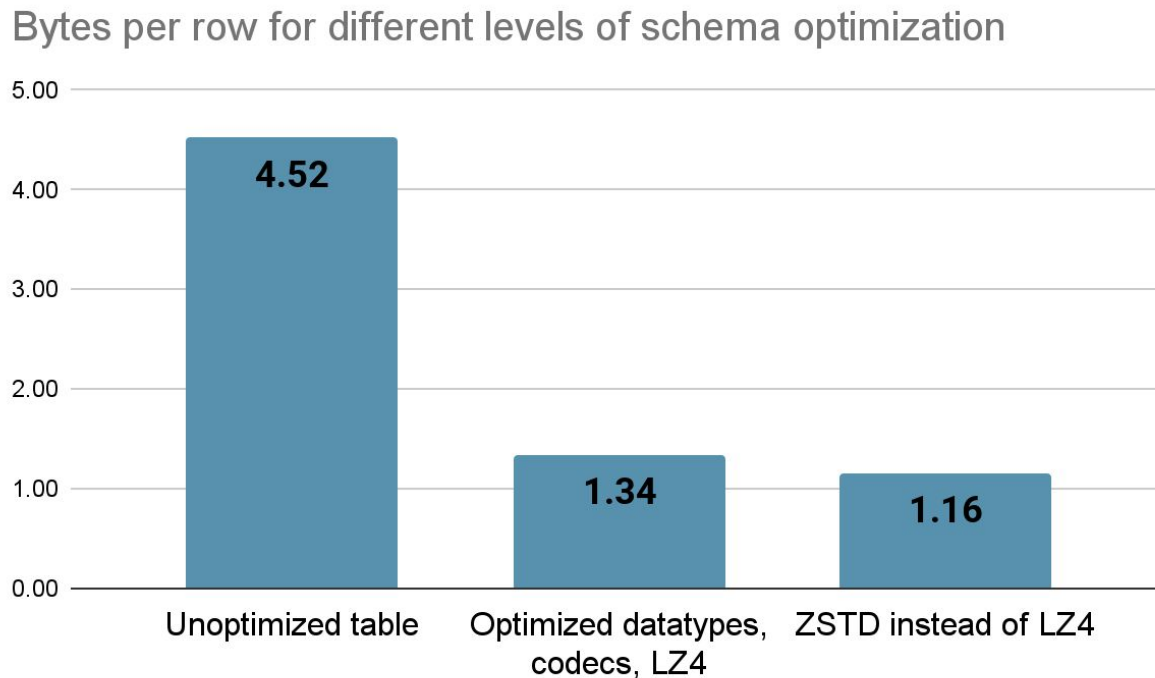
Optimized data types

Codecs + ZSTD compression

ALIAS column to save space

Time ordering to aid compression

# On-disk table size for different schemas



# Learn to love ClickHouse system tables

```
SELECT
    event_time,
    type,
    query_duration_ms / 1000 AS duration,
    read_rows,
    read_bytes,
    result_rows,
    formatReadableSize(memory_usage) AS memory,
    query
FROM system.query_log
WHERE (user = 'test') AND (type = 'QueryFinish')
ORDER BY event_time DESC
LIMIT 50
```



# Analyze queries and make them more efficient

0.84 sec  
1.6 KB RAM

```
SELECT Carrier,  
       avg(DepDelay) AS Delay  
FROM ontime  
GROUP BY Carrier  
ORDER BY Delay DESC  
LIMIT 50
```

Simple aggregate, short  
GROUP BY key with few values

3.4 sec  
2.4 GB RAM

```
SELECT Carrier, FlightDate,  
       avg(DepDelay) AS Delay,  
       uniqExact(TailNum) AS Aircraft  
FROM ontime  
GROUP BY Carrier, FlightDate  
ORDER BY Delay DESC  
LIMIT 50
```

More complex aggregates, longer  
GROUP BY with more values

# How much can we save?

Tune apps to limit resource usage

Savings on infra:  
Up to 90%



# Use TTLs to limit data growth

## TTLs can time out rows

```
CREATE TABLE default.web_events_with_ttl_2 (  
    `time` DateTime,  
    . . .  
    `float_value` Float32  
)  
ENGINE = MergeTree  
PARTITION BY toYYYYMM(time)  
ORDER BY (user_id, toStartOfDay(time), session_id, time)  
TTL time + INTERVAL 12 MONTH DELETE
```

## TTLs can also move, aggregate, and recompress data

```
CREATE TABLE default.web_events_with_ttl_2 (  
    `time` DateTime,  
    . . .  
    `float_value` Float32  
)  
ENGINE = MergeTree  
PARTITION BY toYYYYMM(time)  
ORDER BY (user id, toStartOfDay(time), session id, time)  
TTL time + INTERVAL 1 MONTH RECOMPRESS CODEC (ZSTD(1)),  
    time + INTERVAL 6 MONTH RECOMPRESS CODEC (ZSTD(10)),  
    time + INTERVAL 12 MONTH DELETE
```

## Let's prove it works!

```
SELECT partition, name, rows,  
       data_compressed_bytes AS compressed,  
       data_uncompressed_bytes AS uncompressed  
FROM system.parts  
WHERE (table = 'web_events_with_ttl_2') AND active  
ORDER BY name DESC
```

partition	name	rows	compressed	uncompressed
202304	202304_1_1_0	50000	613930	1388890
202302	202302_2_2_1	50000	327461	1388890
202208	202208_3_3_1	50000	264054	1388890

# How much can we save?

Use TTLs to limit data growth

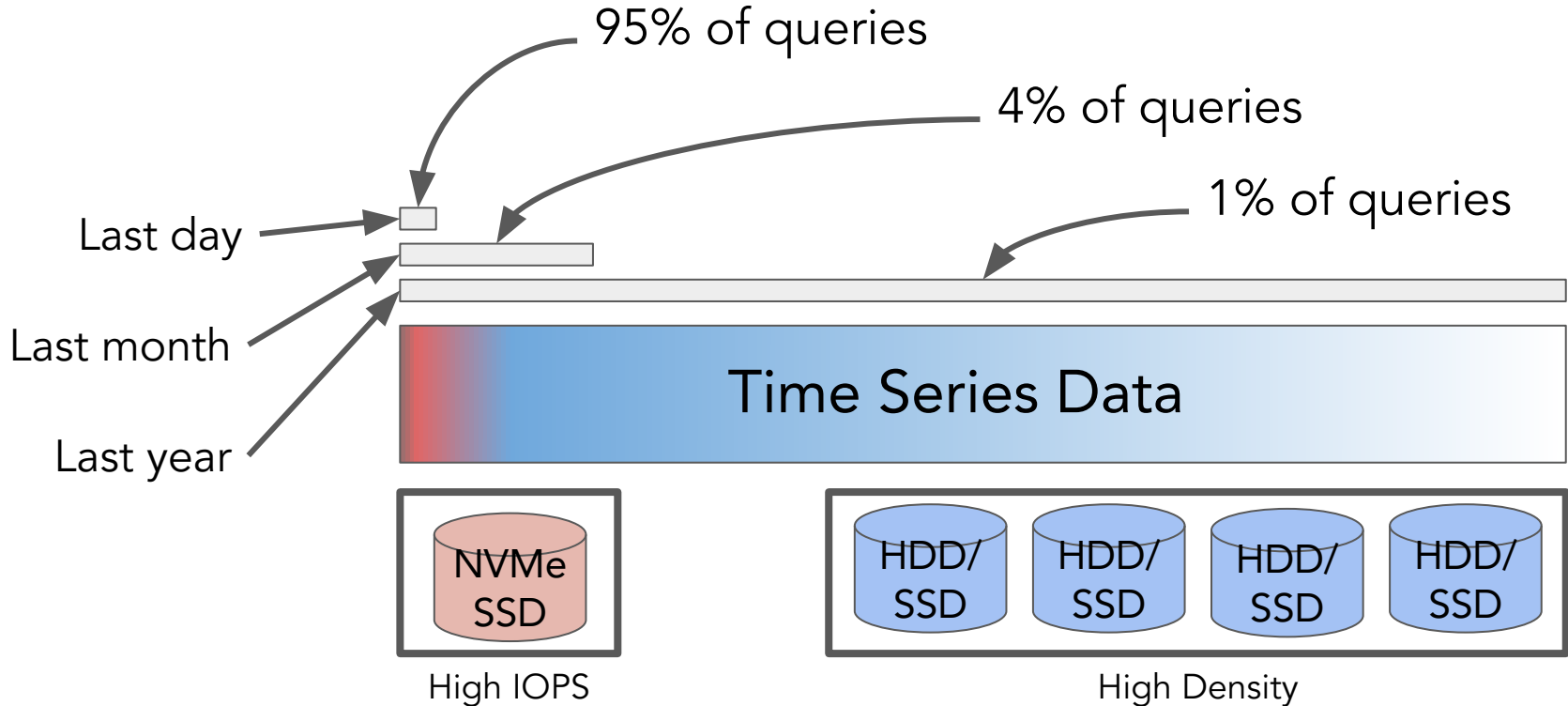
Savings on Storage:  
Up to 20%



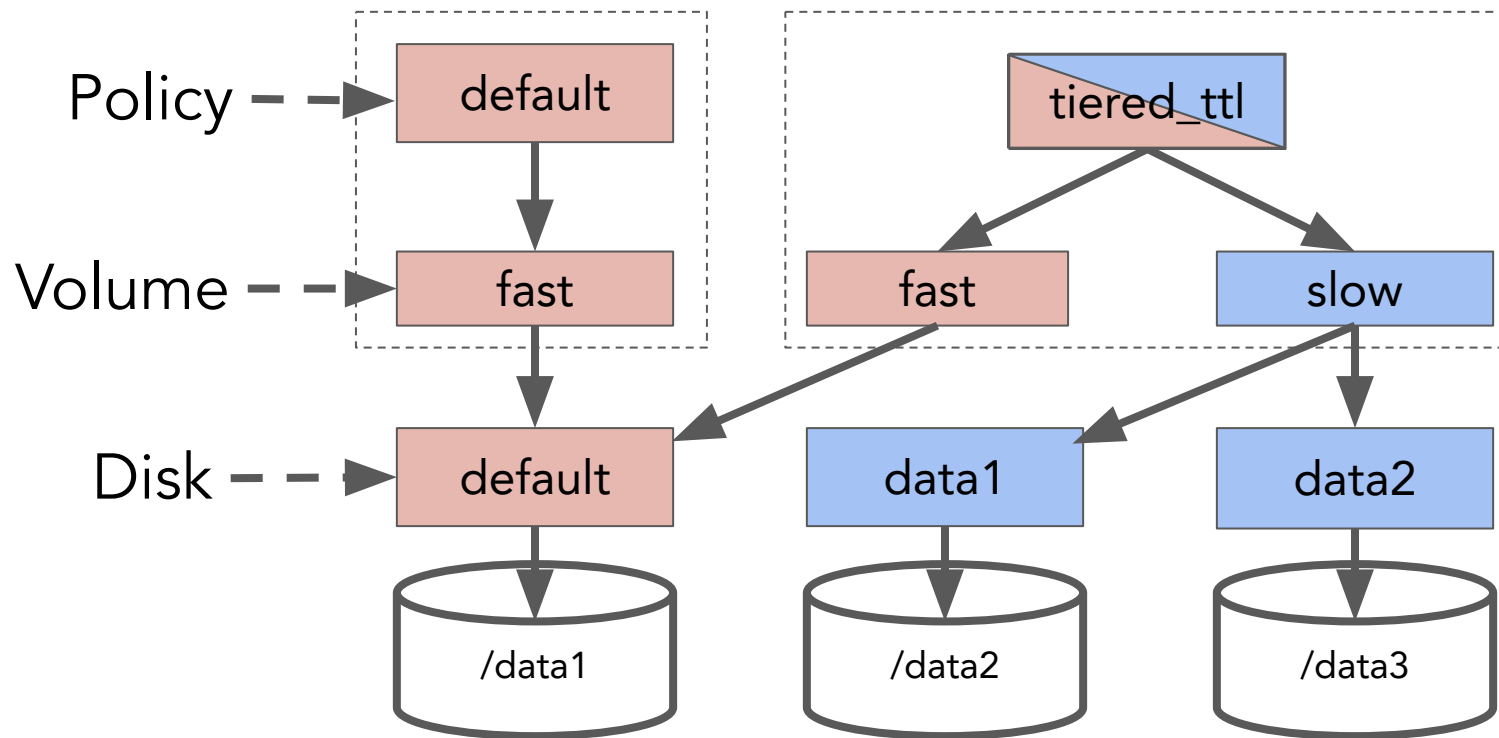
Use tiered  
storage for  
older data



# Tiered storage matches storage cost to access level



# Storage configurations organize devices



# Disks tag lists your devices

```
<clickhouse>
  <storage_configuration>
    <disks>
      <default>
        <keep_free_space_bytes>1024</keep_free_space_bytes>
      </default>
      <data2>
        <path>/data2/clickhouse/</path>
      </data2>
      <data3>
        <path>/data3/clickhouse/</path>
      </data3>
    </disks>
  </storage_configuration>
```

Default disk gets path from config.xml


Storage reserve

Other disks provide a path

# Use simple policies for TTL movement

```
<clickhouse>
<storage_configuration>
  . . .
  <policies>
    <tiered_ttl>
      <volumes>
        <fast>
          <disk>default</disk>
        </fast>
        <slow>
          <disk>data2</disk>
          <disk>data3</disk>
        </slow>
      </volumes>
    </tiered_ttl>
  </policies>
</storage_configuration>
```

Writes go to default if there's no priority specified



TTL clauses move data between volumes

# Manage storage with TTL MOVE and DELETE

```
CREATE TABLE fast_readings (  
    sensor_id Int32 Codec(DoubleDelta, LZ4),  
    time DateTime Codec(DoubleDelta, LZ4),  
    date ALIAS toDate(time),  
    temperature Decimal(5,2) Codec(T64, LZ4)  
) Engine = MergeTree  
PARTITION BY toYYYYMM(time)  
ORDER BY (sensor id, time)  
TTL time + INTERVAL 1 DAY TO VOLUME 'slow',  
    time + INTERVAL 1 YEAR DELETE  
SETTINGS storage_policy = 'tiered ttl'
```

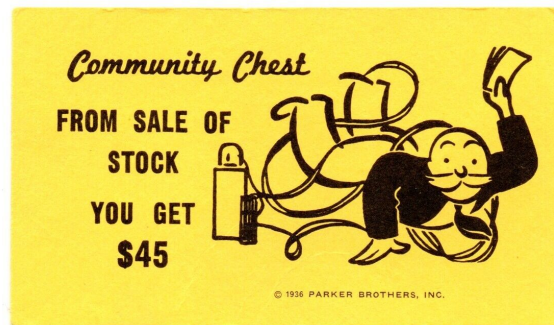
Bonus for even higher savings

Use object  
storage in your  
cold tier

# How much can we save?

Use tiered storage for older data

Savings on Storage:  
Up to 30%

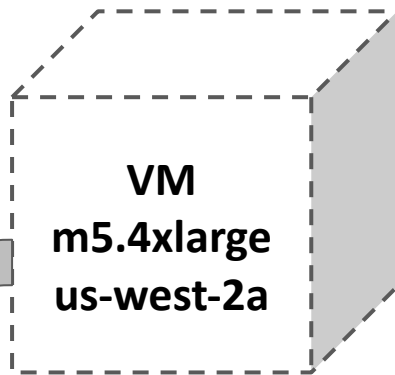


Scale compute  
capacity down  
when not needed

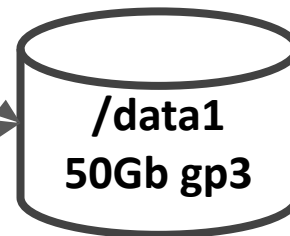


# Re-scaling compute lowers cloud costs dramatically

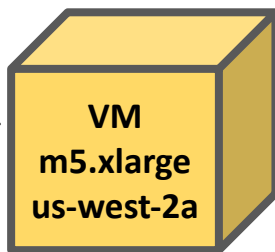
On-demand  
monthly price:  
\$552.64



Network-Attached  
Block Storage

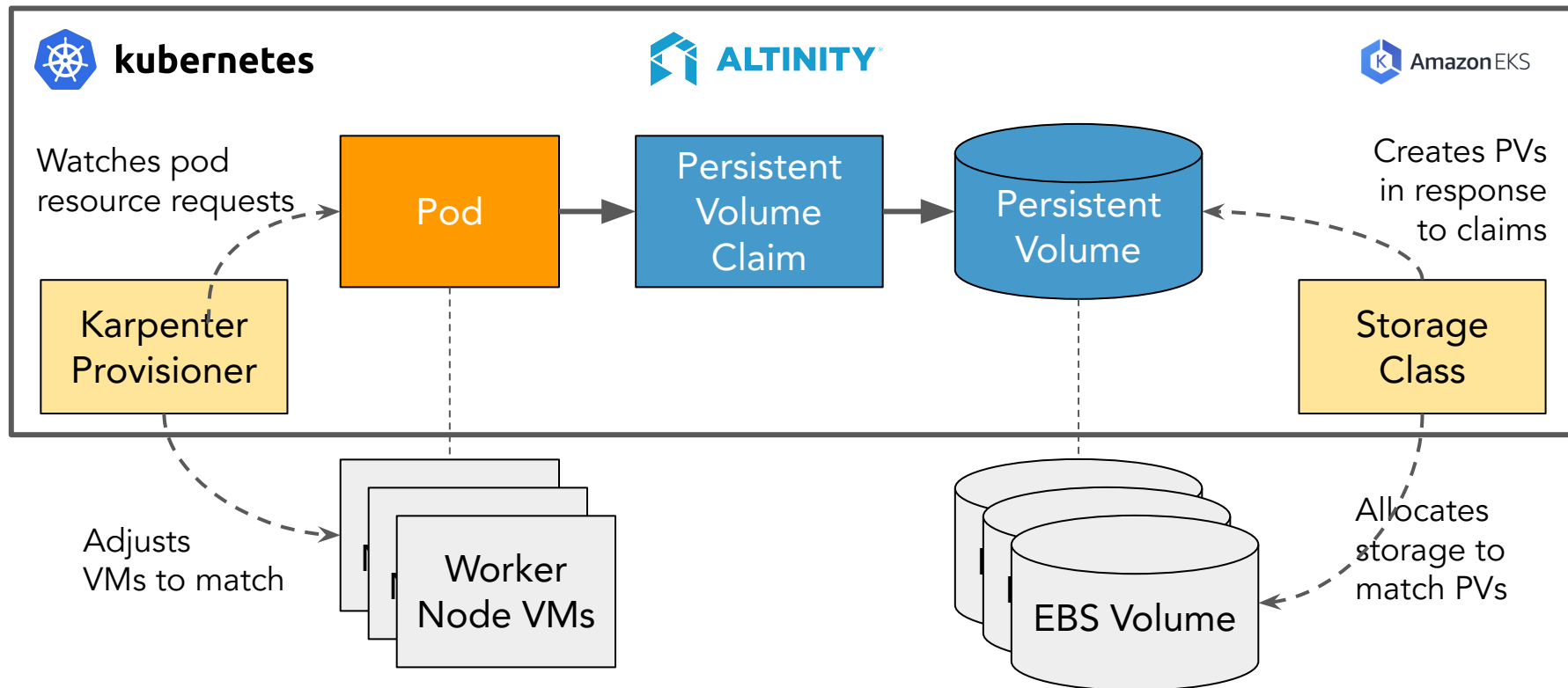


On-demand  
monthly price:  
\$138.24  
(75% cheaper)



On-demand monthly  
price: \$4.00  
(Same)

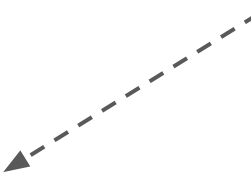
# Kubernetes + Altinity operator makes rescaling easy



# Use pod templates to specify replica properties

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "prod"
spec:
  configuration:
    clusters:
      - name: "ch"
        layout:
          shardsCount: 1
          replicasCount: 1
        templates:
          podTemplate: clickhouse-zone-2a
```

Template for pod definition



# Node selectors and instance types force pods to nodes

```
podTemplates:
```

```
- name: clickhouse-zone-2a
```

```
spec:
```

```
  containers:
```

```
  - name: clickhouse
```

```
    image: altinity/clickhouse-server:22.8.15.25.altinitystable
```

```
  nodeSelector:
```

```
    node.kubernetes.io/instance-type: m5.xlarge
```

```
  zone:
```

```
    key: topology.kubernetes.io/zone
```

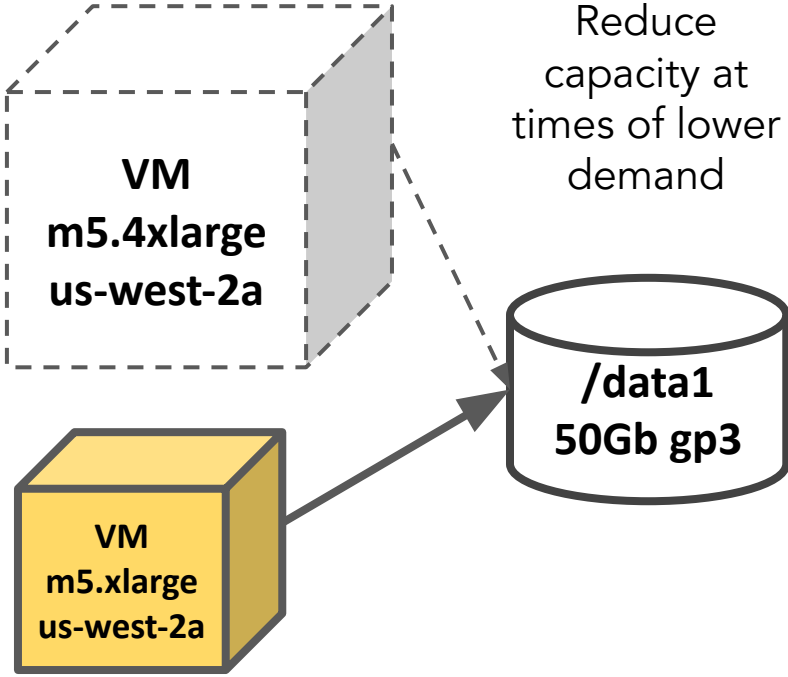
```
    values:
```

```
    - us-west-2a
```

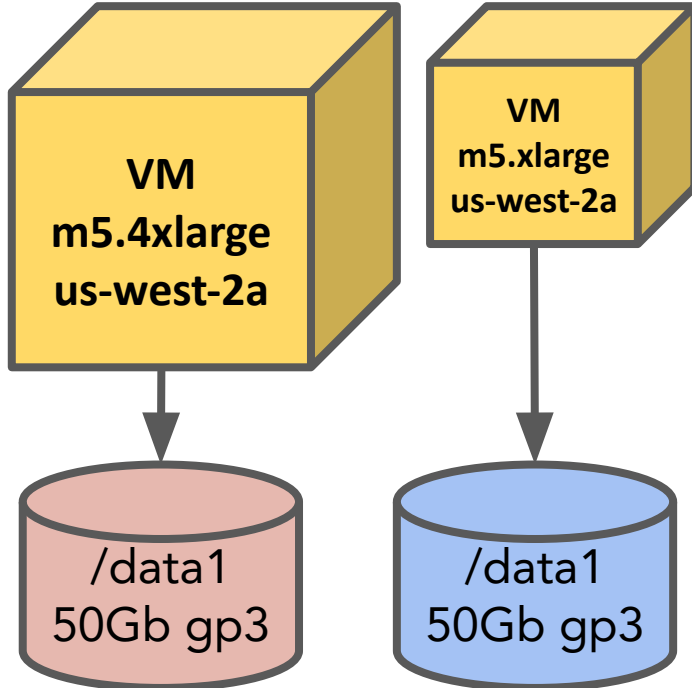
Requires a node with  
m5.xlarge VM type

Requires a node in zone  
us-west-2a

# Patterns for using compute rescaling

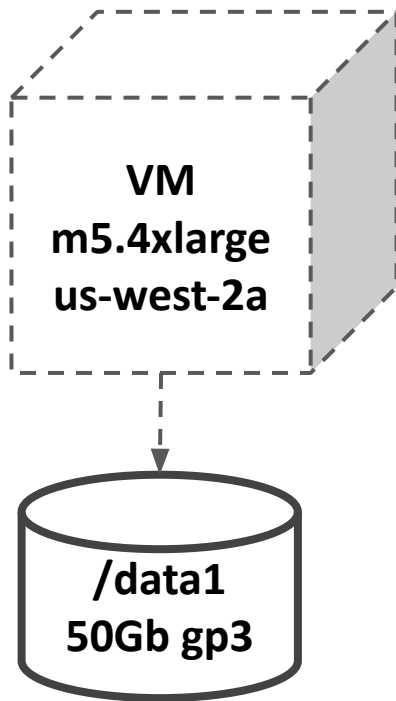


Reduce compute for cold data



# You can also turn off compute completely!

Turn off idle  
development  
hosts



Kubernetes hack:  
edit replica set  
replicas to 0

# How much can we save?

Scale compute capacity down when not needed

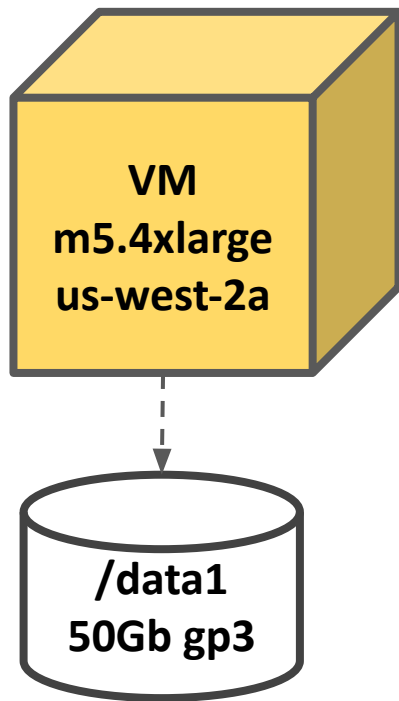
Savings on Compute:  
Up to 50%



# Use Cheaper Resources



# Use cheaper storage types

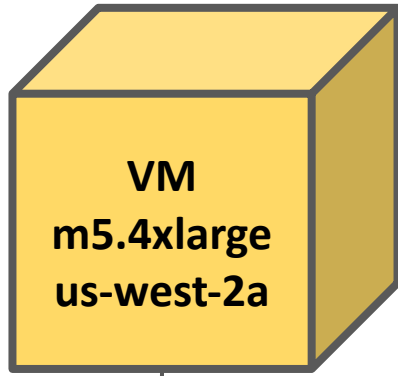


AWS gp3  
storage is 20%  
cheaper than  
gp2

Operator command  
to select storage

```
volumeClaimTemplates:  
- name: storage  
  # Do not delete PVC  
  reclaimPolicy: Retain  
  spec:  
    storageClassName: gp3  
    accessModes:  
      - ReadWriteOnce  
    resources:  
      requests:  
        storage: 50Gi
```

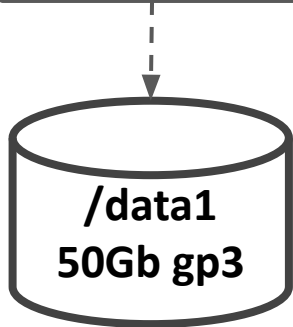
# Use savings plans to reduce compute costs



On-demand monthly price:  
\$548.64

Monthly price for one-year savings plan, no pre-pay:  
\$406.08 (27% cheaper)

Price for three-year savings plan, full-prepay:  
\$9276.84  
(=\$257.69 per month, 54% cheaper)



# Use AWS Graviton (ARM) instead of Intel/AMD instances

Star Schema Benchmark  
ClickHouse version 23.2

Cheapest!

Cheaper and Faster

	m6i.xlarge	m6g.xlarge	m7g.xlarge
SSB Benchmark time	4.551	5.868	3.865
<b>Normalized time</b>	<b>1.175</b>	<b>1.518</b>	<b>1</b>
Us-east-1 hourly cost	0.1920	0.1540	0.1632
<b>Normalized cost</b>	<b>1.2468</b>	<b>1</b>	<b>1.0597</b>

20% less

15% less

<https://altinity.com/blog/ultra-fast-aws-graviton-instances-in-altinity-cloud>

# Use managed hosting instead of public cloud

Up to 90%  
cheaper than  
AWS EC2  
on-demand  
prices

Language: English VAT: DE 19% Community About us Support Login

**HETZNER** Dedicated Cloud Web & Managed Colocation Storage Services

## AX-LINE SERVERS IN THE FAST LANE WITH ZEN ARCHITECTURE HIGH PERFORMANCE AMD RYZEN PROCESSORS

Rapid and parallel processing with powerful multi-core performance.  
Perfect for when you need multiple computing cores.

[Homepage](#) > AX Servers

Sales Chat

no setup fee	no setup fee	++ NEW ++	++ NEW ++	
AX41	AX41-NVME	AX52	AX102	AX161
from € 44.39 monthly incl. 19 % VAT once-off setup: € 46.41 € 0.00	from € 44.39 monthly incl. 19 % VAT once-off setup: € 46.41 € 0.00	from € 70.21 monthly incl. 19 % VAT once-off setup: € 46.41	€ 129.71 monthly incl. 19 % VAT once-off setup: € 46.41	from € 142.56 monthly incl. 19 % VAT once-off setup: € 94.01
Locations: DE +	Locations: DE +	Locations: DE +	Location: DE	Locations: DE +
AX Servers scores ** 🏆 📊	AX Servers scores ** 🏆 📊	AX Servers scores ** 🏆 📊	AX Servers scores ** 🏆 📊	AX Servers scores ** 🏆 📊
<a href="#">CONFIGURE</a>	<a href="#">CONFIGURE</a>	<a href="#">CONFIGURE</a>	<a href="#">CONFIGURE</a>	<a href="#">CONFIGURE</a>
<a href="#">Details &gt;</a>	<a href="#">Details &gt;</a>	<a href="#">Details &gt;</a>	<a href="#">Details &gt;</a>	<a href="#">Details &gt;</a>

AMD

# How much can we save?

## Use Cheaper Resources

Savings on Infra:  
15 to 90%



# Conclusion

# A wallet-sized list of ways to be a ClickHouse cheapskate

Short-term Savings	Time-honored cheapskate practice	Long-term impact
100%	Develop on laptop with 100% open source software	●
Up to 90%	Tune apps to limit resource usage	● ● ●
15% to 90%	Use cheaper resources	● ● ●
Up to 50%	Scale down compute capacity when not needed	● ●
Up to 30%*	Use tiered storage for older data (*bonus for S3)	● ● ●
Up to 20%	Use TTLs to limit data growth	●

# More information!

- Altinity YouTube channel
  - [Tips and Tricks Every ClickHouse User Should Know](#)
  - [A Day in the Life of a ClickHouse Query](#)
- Altinity Blog – <https://altinity.com/blog>
- ClickHouse Documentation – <https://clickhouse.com/docs/en/intro>



# You are now a ClickHouse Cheapskate! Questions?

[Altinity.Cloud](https://altinity.com)

[Altinity Support](https://altinity.com)

[Altinity Stable Builds](https://altinity.com)

[rhodges@altinity.com](mailto:rhodges@altinity.com)

<https://altinity.com>

