

ClickHouse **Defense** Against the **Dark Arts**

Intro to Security and Privacy

Altinity Engineering Team

Presenter Bios and Altinity Introduction

Robert Hodges - Altinity CEO

30+ years on DBMS plus
virtualization and security.
ClickHouse is DBMS #20

Alexander Zaitsev - Altinity CTO

Altinity founder with decades
of expertise on petabyte-scale
analytic systems



The #1 enterprise ClickHouse provider. Now offering [Altinity.Cloud](#)

Major committer and community sponsor for ClickHouse in US/EU

Altinity contributions on security and privacy

- Completed work
 - Log query masking (implementation and testing)
 - AES encryption functions (implementation and testing)
 - RBAC (testing)
 - LDAP user authentication (implementation and testing)
 - LDAP role mapping (implementation and testing)
- In-flight work
 - Server-side Kerberos support (implementation and testing)
 - BoringSSL encryption (testing)
 - Lightweight DELETE/UPDATE (implementation and testing)
 - Audit trail (implementation and testing)
- Roadmap
 - Trusted builds, FedRAMP/FIPS compliance, transparent data encryption

Introducing ClickHouse

ClickHouse is an open source data warehouse

Single binary

Understands SQL

Runs on bare metal to cloud

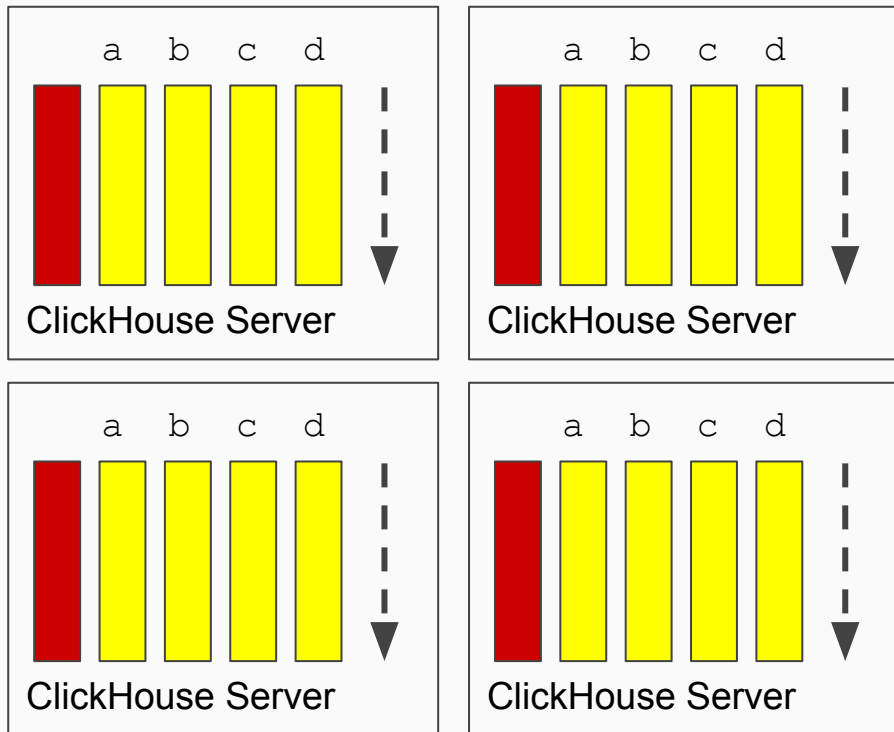
Stores data in columns

Parallel and vectorized execution

Scales to many petabytes

Is Open source (Apache 2.0)

And it's really fast!



But...It's not enough to be fast



Security

**Protecting ClickHouse and
data within it from internal
and external attacks**



Privacy

**Building applications that
comply with standards for
protecting user data**

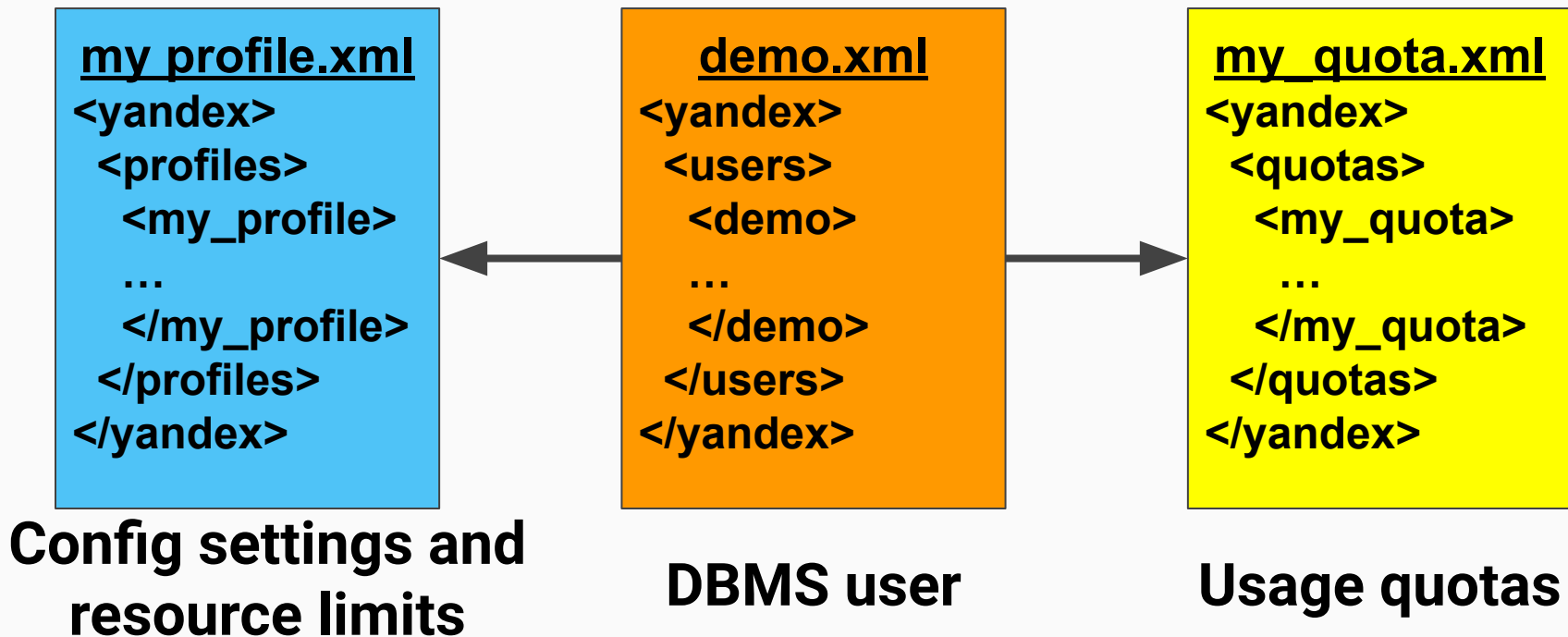
Securing ClickHouse Servers

Topics for securing servers

- Setting up users
- Authorizing access to resources
- Encrypting in-flight communications
- Encrypting data at rest
- Preventing data leakage
- Securing your ClickHouse host

“Classic” user definition with XML

/etc/clickhouse-server/users.d/



Defining a root user

```
<users>
  <root>
    <password_sha256_hex>2bb80...7a25b</password_sha256_hex>
    <networks>
      <ip>127.0.0.1</ip>
    </networks>
    <profile>default</profile>
    <quota>default</quota>
    <access_management>1</access_management>
  </root>
</users>
```

**Localhost login
only**

**Can create users
and grant rights
with RBAC**

Tips for defining users

Generating passwords -- don't forget '-n'

```
echo -n "secret" | sha256sum | tr -d '-'
```

Network masks:

<networks>

<ip>127.0.0.1</ip>

Localhost only

<ip>192.168.128.1/24</ip>

Allow from subnet

<host>logos2</host>

Allow from host

<host_regexp>^logos[1234]\$</host_regexp>

</networks>

Allow from logos1,
logos2, logos3, logos4

Wouldn't this be better in SQL?

Yes! Here's what you can do as of ClickHouse version 20.5

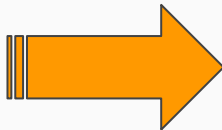
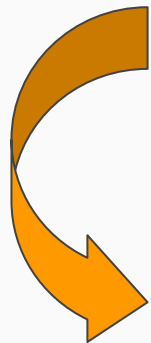
```
-- Create a read-only user that can only access default db.  
CREATE USER IF NOT EXISTS read_only  
    IDENTIFIED WITH SHA256_PASSWORD BY 'secret'  
    HOST IP '192.168.128.1/24' SETTINGS readonly=1;  
REVOKE ALL ON *.* FROM read_only;  
CREATE ROLE select_on_default;  
GRANT SELECT ON default.* TO select_on_default;  
GRANT select_on_default TO read_only;
```

RBAC grants/revokes are very granular

- Tables: CREATE, INSERT, ALTER, DELETE, SELECT, TRUNCATE, OPTIMIZE, DROP...
- Database: CREATE, DROP, SHOW
- Access management: USER, ROLE, POLICY, ROW POLICY, QUOTA, PROFILE, ...
- SYSTEM commands: SHUTDOWN, DROP CACHE, RELOAD, ...

What happens when you create a user?

```
CREATE USER IF NOT EXISTS readonly  
  IDENTIFIED WITH SHA256_PASSWORD BY 'secret'  
  HOST IP '192.168.128.1/24'  
  SETTINGS readonly=1;
```



**/var/lib/clickhouse/access/
5b49c973-124c-4cae-f1e8-14e0644abe91.sql**

```
ATTACH USER read_only IDENTIFIED  
WITH sha256_hash BY  
'2BB80...7A25B' HOST IP  
'192.168.128.0/24' SETTINGS  
readonly = 1;
```

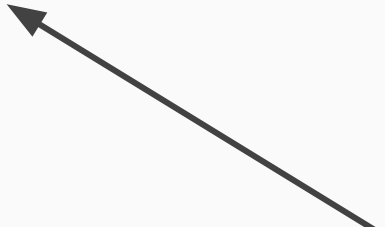
LDAP authentication is available since 20.8

```
CREATE USER IF NOT EXISTS ldap_user  
  IDENTIFIED WITH ldap_server BY 'ldap_local'  
  HOST ANY
```

**LDAP server name in
config.xml**

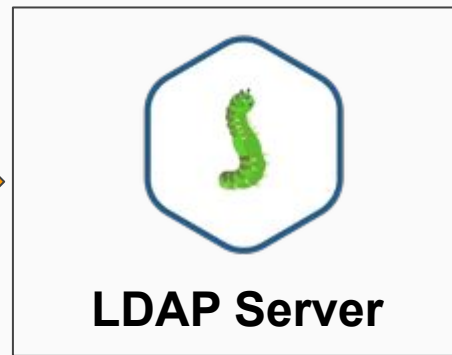
A yellow box containing the text "LDAP server name in config.xml" has a black arrow pointing from its bottom-left corner to the "ldap_server" keyword in the SQL command above.

**Note: keyword will
change to 'ldap' when
Kerberos support is
added**

A yellow box containing the note "Note: keyword will change to 'ldap' when Kerberos support is added" has a black arrow pointing from its top-right corner to the "ldap_server" keyword in the SQL command above.

LDAP authentication flow

```
clickhouse-client  
--user=ldap_user  
--password=secret
```

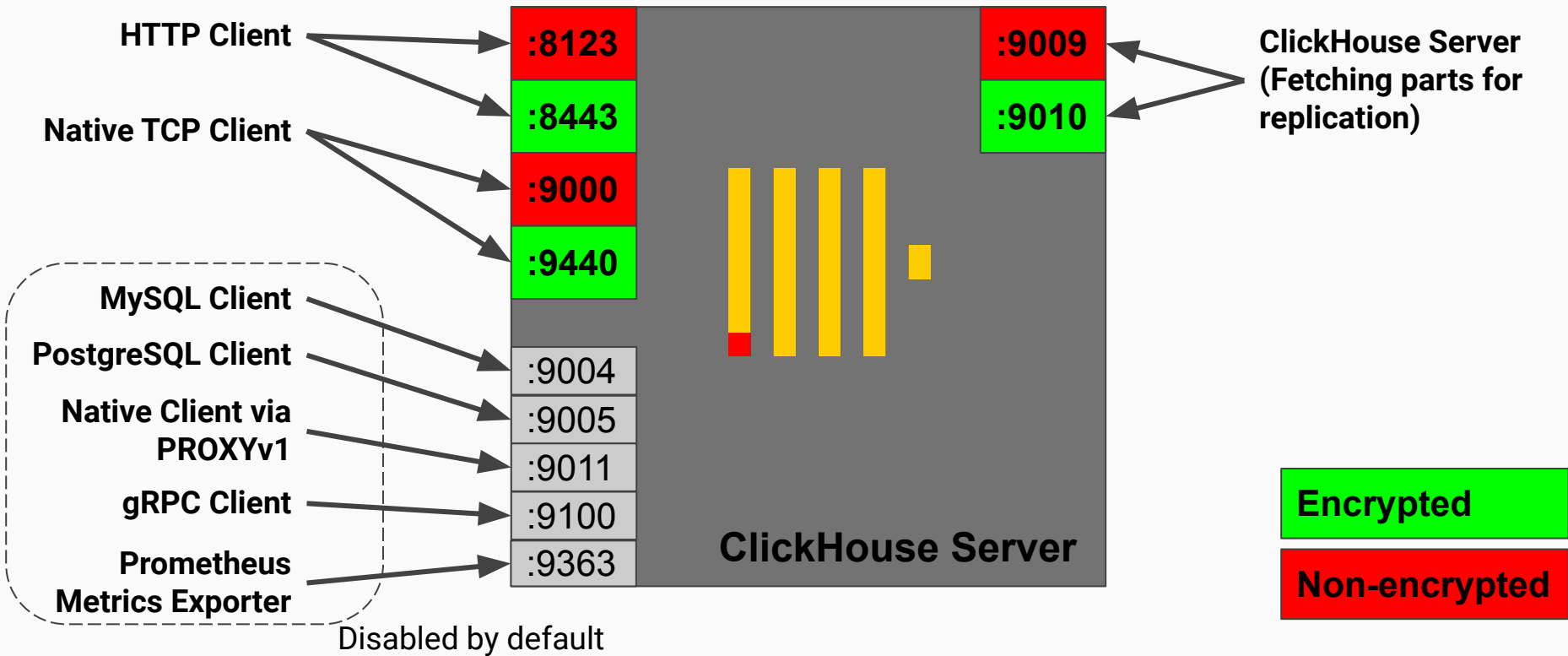


config.xml

```
<yandex>  
  <ldap_servers>  
    <ldap_local>  
      <host>ldap-01</host>  
      ...  
    </ldap_local> ...  
  </ldap_servers>  
</yandex>
```

```
uid=ldap_user,ou=users,dc=example,dc=com  
=>  
userPassword: secret
```


In-flight connections: attack surfaces



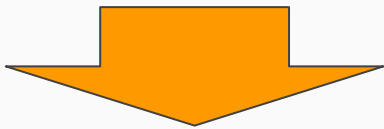
Steps to protect in-flight connections

1. Turn off all unused ports
2. Enable TLS encryption for native TCP and HTTP clients
 - Disable unencrypted ports
3. For clusters:
 - Switch to TLS on interserver communications
 - Define user for remote calls

Turning off unused ports

/etc/clickhouse-server/config.xml

```
...  
<mysql_port>9004</mysql_port>  
...  
<interserver_http_port>9009</interserver_http_port>
```



```
...  
<!-- <mysql_port>9004</mysql_port> -->  
...  
<!-- <interserver_http_port>9009</interserver_http_port> -->
```

Enabling TLS client connections

/etc/clickhouse-server/config.xml


```
<!-- <http_port>8123</http_port> -->
<!-- <tcp_port>9000</tcp_port> -->
...
<https_port>8443</https_port>
<tcp_port_secure>9440</tcp_port_secure>
...
<openSSL>
  <server>
    <certificateFile>/etc/clickhouse-server/server.crt</certificateFile>
    <privateKeyFile>/etc/clickhouse-server/server.key</privateKeyFile>
    <dhParamsFile>/etc/clickhouse-server/dhparam.pem</dhParamsFile>
    ...
  
```

Options for ClickHouse server certificates



**Certificate from
a public CA like
Let's Encrypt**

Ideal for external services
or services with clients
you don't control



**Certificate from
your own
internal CA**

Ideal for internal services
where you control the
clients and their operating
environment



**Self-signed
certificate**

Useful for testing. **Never
use this approach for real
data.**

For more information: <https://altinity.com/blog/2019/3/5/clickhouse-networking-part-2>

Tips and gotchas for TLS encryption

1. Connection credentials are transmitted in the clear unless you enable TLS!
2. Certificates from non-public CAs require different steps for each app type

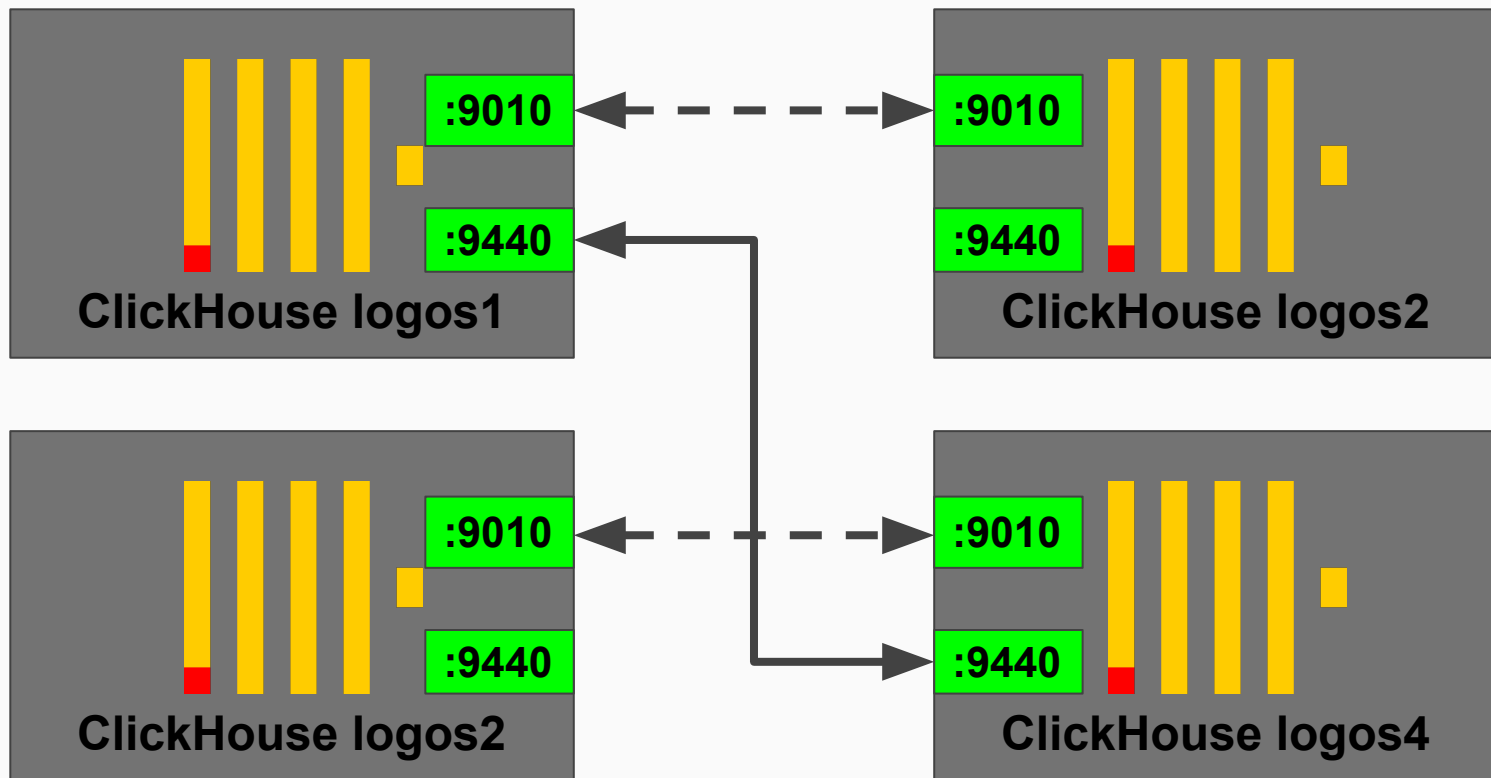
```
export NODE_EXTRA_CA_CERTS=/etc/ssl/certs/my.root.ca.pem
node my-tls-enabled-client.js
```

3. Internal root CAs require proper security hygiene to maintain
4. Check ports to make sure you enabled/disabled correctly

```
sudo netstat -lntp |grep clickhouse
```

tcp6	0	0	:::8443	:::*	LISTEN	7768/clickhouse-ser
tcp6	0	0	:::9440	:::*	LISTEN	7768/clickhouse-ser
tcp6	0	0	:::9010	:::*	LISTEN	7768/clickhouse-ser

Encrypting ClickHouse cluster traffic



Create a user for distributed queries

```
CREATE USER IF NOT EXISTS internal ON CLUSTER 'my_cluster'  
  IDENTIFIED WITH NO_PASSWORD  
  HOST REGEXP '^logos[1234]$'
```



**Network mask restricts
user to within cluster**

Enable TLS for interserver replication

/etc/clickhouse-server/config.xml

```
<yandex>
  <!-- <interserver_http_port>9009</interserver_http_port> -->

  <interserver_https_port>9010</interserver_https_port> -->
  ...
  <interserver_http_credentials>
    <user>internal</user>
    <password></password>
  </interserver_http_credentials>
```

**Secure
interserver port**

Enable interserver auth

TLS connections for distributed queries

/etc/clickhouse-server/config.d/remote_servers.xml

```
<yandex>
  <remote_servers>
    <my_cluster>
      <shard>
        <internal_replication>true</internal_replication>
        <replica>
          <host>logos1</host>
          <port>9440</port>
          <secure>1</secure>
          <user>internal</port>
        </replica>
        ...
      </shard>
```

Secure port and TLS

User with no password

New in ClickHouse 20.10+ (preferred way)

/etc/clickhouse-server/config.d/remote_servers.xml

```
<yandex>
  <remote_servers>
    <my_cluster>
      <shard>
        <secret>shared secret text</secret>
        <internal_replication>true</internal_replication>
        <replica>
          <host>logos1</host>
          <port>9440</port>
          <secure>1</secure>
        </replica>
        ...
      </shard>
    
```

Use initial user across servers; authenticate with secret

Secure port and TLS (as before)

(<https://github.com/ClickHouse/ClickHouse/pull/13156>)

Encryption at-rest, option 1: file system

File system encryption is the simplest path to global at-rest encryption.

Some of our favorite options:

1. LUKS (Linux Unified Key Setup) -- Encrypt local volume using DMCCrypt
2. Cloud block storage encryption -- Use public cloud automatic encryption
 - a. Example: Amazon EBS encryption
3. Kubernetes storage provider encryption -- Enable encryption in StorageClass if supported.
 - a. Example: AWS EBS provider encrypted: "true" option

Encryption at-rest, option 2: AES functions

Starting in 20.11 these can be used to protect data at level of individual columns.

```
encrypt(mode, plaintext, key, [iv, aad])  
decrypt(mode, ciphertext, key, [iv, aad])  
aes_encrypt_mysql(mode, plaintext, key, [iv])*  
aes_decrypt_mysql(mode, ciphertext, key, [iv])*
```

Key management is responsibility of applications (for now)

*Compatible with MySQL AES functions

Examples of AES functions in use

```
WITH unhex('658bb26de6f8a069a3520293a572078f') AS key
SELECT hex(encrypt('aes-128-cbc', 'Hello world', key)) AS encrypted
```

encrypted

46924AC12F4915F2EEF3170B81A1167E

```
WITH unhex('658bb26de6f8a069a3520293a572078f') AS key
SELECT decrypt('aes-128-cbc',
  unhex('46924AC12F4915F2EEF3170B81A1167E'), key) AS plaintext
```

plaintext

Hello world

Important pro tip: test performance!!!

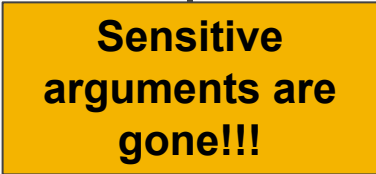
Avoiding data leakage with query masking

ClickHouse log entry for our query examples:

```
2021.01.26 19:11:23.526691 [ 1652 ] {4e196dfa-dd65-4cba-983b-d6bb2c3df7c8}  
<Debug> executeQuery: (from [::ffff:127.0.0.1]:54536, using production  
parser) WITH unhex('658bb26de6f8a069a3520293a572078f') AS key SELECT  
decrypt(???), key) AS plaintext
```



**Hey, that's an
AES key!**



**Sensitive
arguments are
gone!!!**

How to make queries “disappear” from logs

/etc/clickhouse-server/config.xml

```
<query_masking_rules>
  <rule>
    <name>hide encrypt/decrypt arguments</name>
    <regexp>
      ((?:aes_)?(?:encrypt|decrypt) (?:_mysql)?) \s* \ ( \s* (?: ' (?:
\\ ' | .) + ' | .*?) \s* \ )
    </regexp>
    <!-- or more secure, but also more invasive:
      (aes_\w+) \s* \ (. * \ )
    -->
    <replace>\1(???)</replace>
  </rule>
</query_masking_rules>
```


Final tips: host-level security

1. ClickHouse runs as clickhouse user (root access not required)
2. Protect directories containing data and credentials
 - a. /etc/clickhouse-server -- Credentials
 - b. /var/lib/clickhouse -- Data and (new!) credentials
 - c. /var/log/clickhouse-server -- Logs, SQL queries may be exposed
3. Protect the network around ClickHouse
 - a. Load balancers, firewalls
 - b. Make server network non-routable to outsiders as far as possible

Building Privacy-aware Applications

Multi-tenancy models

Model	How it works	Shared resources
Dedicated Installation	Separate ClickHouse per tenant	Network?
Dedicated Cluster	Set of shards and replicas per tenant	ZooKeeper, configuration files, network
Dedicated Databases	Separate database per tenant	ZooKeeper, ClickHouse hosts, configuration, network
Dedicated Tables	Separate table(s) per tenant	ZooKeeper, ClickHouse hosts, configuration, network
Shared Tables	Tenant data inhabits shared tables. Each row has a tenant key	ZooKeeper, ClickHouse hosts, tables, configuration, network

Restricting user access in shared clusters

Using configuration in users.xml:

```
<yandex>
  <users>
    <demo>
      <allow_databases>
        <database>demo</database>
      </allow_databases>
    </demo>
  </users>
```


Restrict access to DB



Using RBAC:

```
GRANT SELECT ON demo.* TO demo
GRANT SELECT ON customers.demo2_table TO demo2
```

Restrict access to a table



Row-level access control

Using configuration in users.xml:

```
<yandex>
  <users>
    <demo>
      <databases>
        <default>
          <my_table>
            <filter>user_id='demo'</filter>
          </my_table>
        </default>
      </databases>
    </demo>
  </users>
```

Restrict access to rows



Using RBAC:

```
CREATE ROW POLICY filter ON default.my_table FOR SELECT USING user_id='demo' TO demo
```

DELETE tenant data efficiently

Efficient if partition is dropped fully

```
ALTER TABLE DROP PARTITION
```

- Database or table per tenant -- easy to drop, but does not scale well beyond dozens or low hundreds
- Partition per tenant -- easy to drop, scales to 1000s, row-level security required
- All shared -- expensive to drop (`ALTER TABLE DELETE`), scales well
 - `ALTER TABLE DELETE IN PARTITION`-- available in 20.12
 - Lightweight DELETES in 2021 roadmap

Data retention options

```
CREATE TABLE (  
  ...  
) Engine = MergeTree ...
```

- TTL DELETE toStartOfMonth(date) + interval 1 month -- efficient with monthly partitioning
- TTL DELETE date + ttl_days_to_keep -- efficient with daily partitioning
- TTL DELETE date + dictGet('tenant_ttl', 'ttl_days', tenant_id) -- configure externally
- TTL DELETE WHERE -- arbitrary expression
- ttl_only_drop_parts=1 (default 0) -- make sure no expensive operations
- Column TTLs:

```
customer_id UUID TTL date + interval 1 month
```

Other things to consider in app design

- System.query_log, system.text_log, system.processes access
- Encrypting sensitive data with AES functions
 - “Losing” keys as a way of deleting data
- Constraints on settings:
 - <https://clickhouse.tech/docs/en/operations/settings/constraints-on-settings/>
- Secrets management
 - Currently an application responsibility
 - In roadmap (“Transparent data encryption”)

Wrap-Up

Summary

- ClickHouse security advanced rapidly in 2020
 - RBAC, LDAP authentication, AES encryption
- Rich feature set to secure data in-flight and at-rest
- Privacy is a property of application design
- Major improvements on tap in 2021
 - Already merged: BoringSSL for TLS, LDAP role mapping
 - In progress: Kerberos, lightweight DELETE/UPDATE

Thank you!

**Contact us to
discuss ClickHouse
security needs**

ClickHouse:

[https://github.com/ClickHouse/
ClickHouse](https://github.com/ClickHouse/ClickHouse)

Altinity Blog:

<https://altinity.com/blog>

Contact:

info@altinity.com