User Management in ClickHouse® Databases

The Unabridged Edition

Robert Hodges - Altinity CEO
Alexander Zaitsev - Altinity CTO

Let's make some introductions

Robert Hodges

Database geek with 30+ years on DBMS. Kubernaut since 2018. Day job: Altinity CEO

Alexander Zaitsev

Expert in high scale analytics systems design and implementation. Altinity CTO

ClickHouse support and services including Altinity.Cloud
Authors of Altinity Kubernetes Operator for ClickHouse
and other open source projects

ClickHouse is a real-time analytic database

Understands SQL

Runs on bare metal to cloud

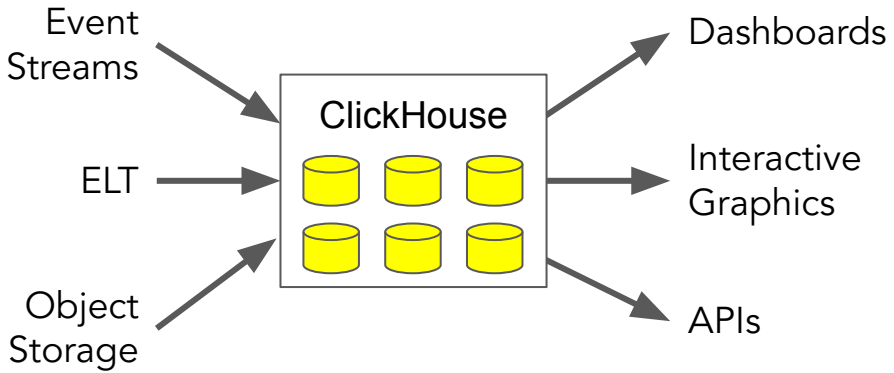
Shared nothing architecture

Stores data in columns

Parallel and vectorized execution

Scales to many petabytes

Is Open source (Apache 2.0)



It's a popular engine for real-time analytics

Developer intro to user management in ClickHouse

Where is my user?

```
$ clickhouse-client

ClickHouse client version 24.3.5.46 (official build).
Connecting to localhost:9000 as user default.
Connected to ClickHouse server version 24.3.5.

:) select currentUser()
```

```
┌─currentUser()─┐
│ default       │
└───────────────┘
```

Default user has no password! Be careful!

How do I make my first user? With an XML file!

```
<clickhouse>
  <users>
    <xmluser>
      <access_management>1</access_management>
      <networks><ip>::1</ip></networks>
      <password_sha256_hex>b04e6...0c95</password_sha256_hex>
      <profile>default</profile>
      <quota>default</quota>
    </xmluser>
  </users>
</clickhouse>
```

How do I make hashed passwords?

Use these *exact* commands!

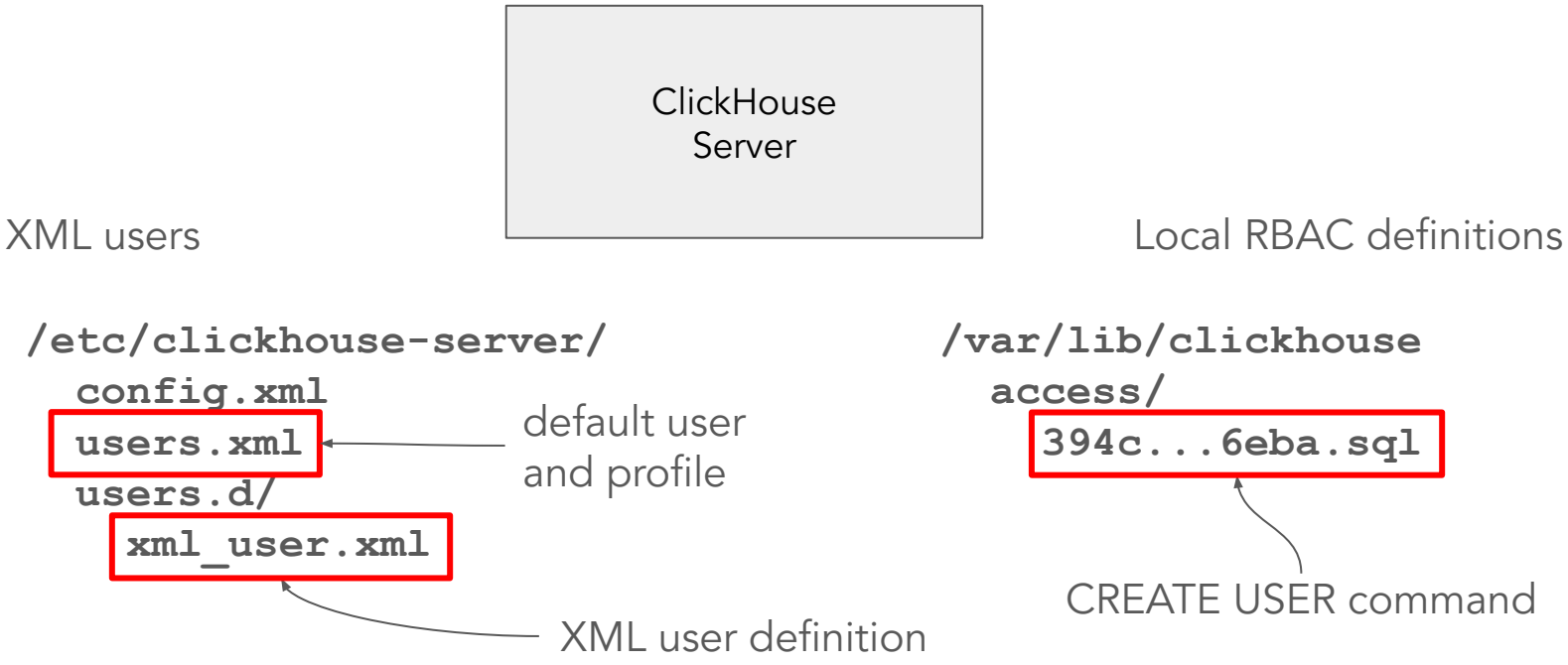
```
# Generate a secure password.
PASSWORD=$(base64 < /dev/urandom | head -c8); \
echo "$PASSWORD"; \
echo -n "$PASSWORD" | sha256sum | tr -d '-'
TPo6F7vT
b04e631478a1a9de206499e4bdc2eae900fc6f994ca49ea908797e62eecc0c95

clickhouse-connect --user=u_sha256 --password=TPo6F7vT
```

CREATE USER command is *much* easier!

```
CREATE USER IF NOT EXISTS rbac_user
  IDENTIFIED WITH sha256_password BY 'topsecret'
  HOST LOCAL
  SETTINGS PROFILE 'default'
```

RBAC = “Role Based Access Control”

Where does ClickHouse store users?



What about network masks?

```
CREATE USER IF NOT EXISTS newuser_any_host
  IDENTIFIED WITH sha256_password BY 'topsecret'
  HOST ANY

CREATE USER IF NOT EXISTS newuser_localhost
  IDENTIFIED WITH sha256_password BY 'topsecret'
  HOST IP 127.0.0.1

CREATE USER IF NOT EXISTS newuser_network
  IDENTIFIED WITH sha256_password BY 'topsecret'
  HOST IP '10.0.0.1/8'
```

What about profile settings?

```
CREATE SETTINGS PROFILE `rmt_profile` SETTINGS
  log_queries = true,
  final = true,
  do_not_merge_across_partitions_select_final = true,
  load_balancing = 'nearest_hostname',
  prefer_localhost_replica = false

CREATE USER rmt_user
  IDENTIFIED WITH sha256_password 'topsecret'
  SETTINGS PROFILE = 'rmt_profile'
```

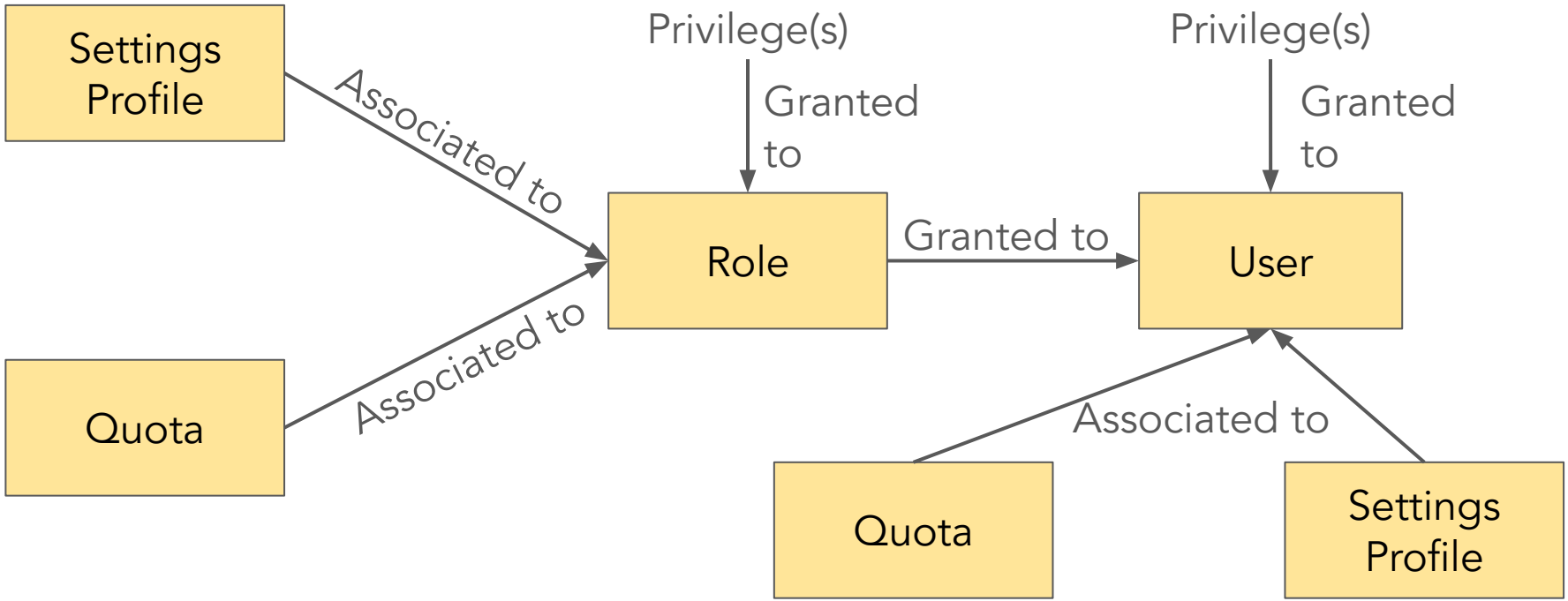
Creating a user on a cluster

```
CREATE USER IF NOT EXISTS rbac_user
   ON CLUSTER '{cluster}'
   IDENTIFIED WITH sha256_password BY 'topsecret'
   HOST LOCAL
   SETTINGS PROFILE 'default'
```

```
┌─host──────────────┬─port─┬─status─┬─error─┬─num_hosts_remaining─┬─...─┐
│ chi-demo-rbac-1-0 │ 9000 │      0 │       │                   1 │   0 │
│ chi-demo-rbac-0-0 │ 9000 │      0 │       │                   0 │   0 │
└───────────────────┴──────┴────────┴───────┴─────────────────────┴─────┘
```

Understanding RBAC in ClickHouse

The RBAC Model: Users, Roles, Profiles, and Quotas



Creating a profile and a role that uses it

```
CREATE SETTINGS PROFILE IF NOT EXISTS u2_profile
SETTINGS
  max_threads = 2 MIN 1 MAX 4,
  max_memory_usage = 10000000 MIN 1000000 MAX 20000000
READONLY
;

CREATE ROLE IF NOT EXISTS u2_role
  SETTINGS PROFILE 'u2_profile'
;
```

Revoking and granting privileges on roles

```
REVOKE ALL FROM u2_role ;
GRANT SHOW DATABASES ON system.* TO u2_role;
GRANT SELECT ON system.* TO u2_role;
GRANT SELECT ON default.* TO u2_role;
```

Creating a resource quota on a role

```
CREATE QUOTA IF NOT EXISTS u2_quota
FOR INTERVAL 30 second
  MAX queries 5,
  MAX result_rows 1000000
TO u2_role;
```

Quota can be set for:

- Queries
- Inserts
- Selects
- Errors
- Result rows/bytes
- Read rows/bytes
- Execution time

=> in a given period of time

Creating a user with the role

```
CREATE USER IF NOT EXISTS u2_user
  IDENTIFIED WITH SHA256_PASSWORD BY 'topsecret'
  HOST ANY
  DEFAULT ROLE u2_role
;
```

Wait, there's more: Row policies!

Data

group	name
1	one
2	two
3	three
4	four
1	five
2	six

Desired Access Rules

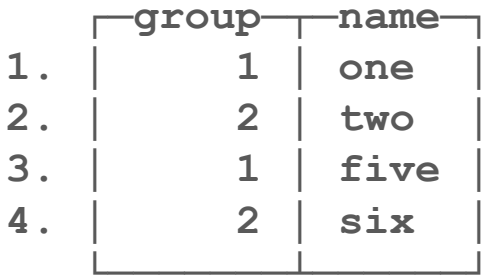
condition	visibility
All users	Can see rows where group = 1
User rp_user2	Can see rows where group < 3
Users with rp_special role	Can see all rows where group < 10

Creating row policies to enforce rules

```
CREATE USER IF NOT EXISTS rp_user0
  NOT IDENTIFIED DEFAULT ROLE rp_basic;
CREATE USER IF NOT EXISTS rp_user1
  NOT IDENTIFIED DEFAULT ROLE rp_special;
CREATE USER IF NOT EXISTS rp_user2
  NOT IDENTIFIED DEFAULT ROLE rp_basic;

CREATE ROW POLICY row_policy0 ON default.rp_example
  USING group=1 AS PERMISSIVE TO ALL;
CREATE ROW POLICY row_policy1 ON default.rp_example
  USING group<10 TO rp_special;
CREATE ROW POLICY row_policy2 ON default.rp_example
  USING group<3 TO rp_user2;
```

Effect of row policies on SELECTs

```
# clickhouse-client --user=rp_user2

:) select * from default.rp_example
```



```
    ┌─group─┬─name─┐
1.  │     1 │ one  │
2.  │     2 │ two  │
3.  │     1 │ five │
4.  │     2 │ six  │
    └───────┴──────┘
```

```
CREATE ROW POLICY row_policy2
  ON default.rp_example
  USING group<3 TO rp_user2;
```

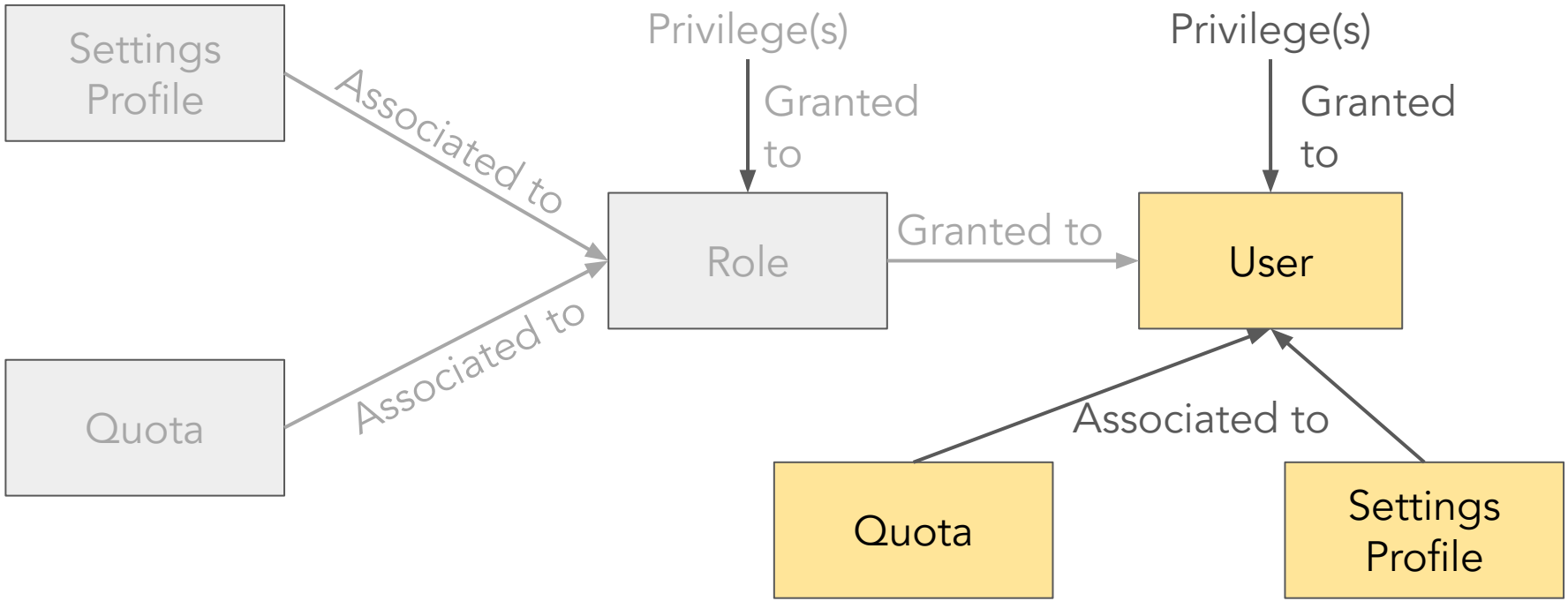
What if there are many tables?

```
CREATE ROW POLICY row_policy0 ON default.*
  USING group=1 AS PERMISSIVE TO ALL;
CREATE ROW POLICY row_policy1 ON default.*
  USING group<10 TO rp_special;
CREATE ROW POLICY row_policy2 ON default.*
  USING group<3 TO rp_user2;
```

Is XML Dead?

More about XML user management

XML user management - No roles!



But you can grant privileges directly to users!

```
<clickhouse>
  <users>
    <xmluser>
      ...
      <grants>
        <query>GRANT SELECT ON default.*</query>
      </grants>
      <databases>
        <default>
            <rp_example>
                <filter>group = 1</filter>
            </rp_example>
        </default>
      </databases>
    </xmluser>
  </users>
</clickhouse>
```

Can only read data from tables in default

This is like a row policy

XML vs RBAC user management compared

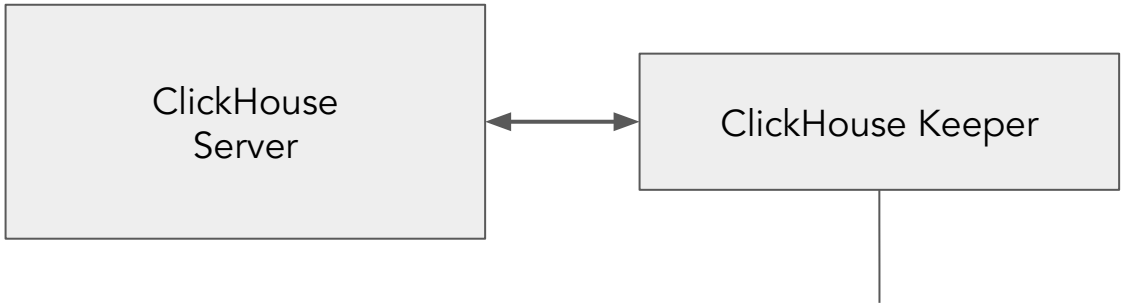
	XML	RBAC
Feature richness	Simpler model; missing roles and row policies	Rich model that supports separation of tenants
Developer friendliness	File-based configuration is simple for deployment	Harder to integrate with IaC (Infrastructure-as-Code)
Ease of deployment	Requires access to file system	Runs over the network
Cluster support	None	ON CLUSTER clause and replicated users

Managing Users in Clusters

Use the ON CLUSTER clause to define RBAC in clusters

```
CREATE ROLE IF NOT EXISTS u2_role ON CLUSTER '{cluster}'
  SETTINGS PROFILE 'u2_profile';

REVOKE ON CLUSTER '{cluster}' ALL FROM u2_role ;
GRANT ON CLUSTER '{cluster}' SHOW DATABASES ON system.* TO u2_role;
GRANT ON CLUSTER '{cluster}' SELECT ON system.* TO u2_role;
GRANT ON CLUSTER '{cluster}' SELECT ON default.* TO u2_role;

CREATE USER IF NOT EXISTS u2_user ON CLUSTER '{cluster}'
  IDENTIFIED WITH SHA256_PASSWORD BY 'topsecret'
  HOST ANY
  DEFAULT ROLE u2_role;
```

There's another approach to storing RBAC



XML users in local storage

```
/etc/clickhouse-server/
  config.d/
    user_directories.xml
  config.xml
  users.xml
  users.d/
    xmluser.xml
```

```
/clickhouse/access/mycluster/
    (Nodes for RBAC data)
```

Replicated users in Keeper

Example of replicated user directory configuration

/etc/clickhouse-server/config.d/user_directories.xml:

```
<clickhouse>
  <user_directories replace="replace">
    <users_xml>
      <path>/etc/clickhouse-server/users.xml</path>
    </users_xml>
    <replicated>
      <zookeeper_path>/clickhouse/mycluster/access/</zookeeper_path>
    </replicated>
  </user_directories>
</clickhouse>
```

RBAC commands are now much simpler

```
CREATE ROLE IF NOT EXISTS u2_role
  SETTINGS PROFILE 'u2_profile';

REVOKE ALL FROM u2_role ;
GRANT SHOW DATABASES ON system.* TO u2_role;
GRANT SELECT ON system.* TO u2_role;
GRANT SELECT ON default.* TO u2_role;

CREATE USER IF NOT EXISTS u2_user
  IDENTIFIED WITH SHA256_PASSWORD BY 'topsecret'
  HOST ANY
  DEFAULT ROLE u2_role;
```

RBAC commands replicate automatically to new shards and replicas!!

And that's not all folks…

LDAP

Can operate as password store or user directory with usres and roles

Certificate-Based Auth

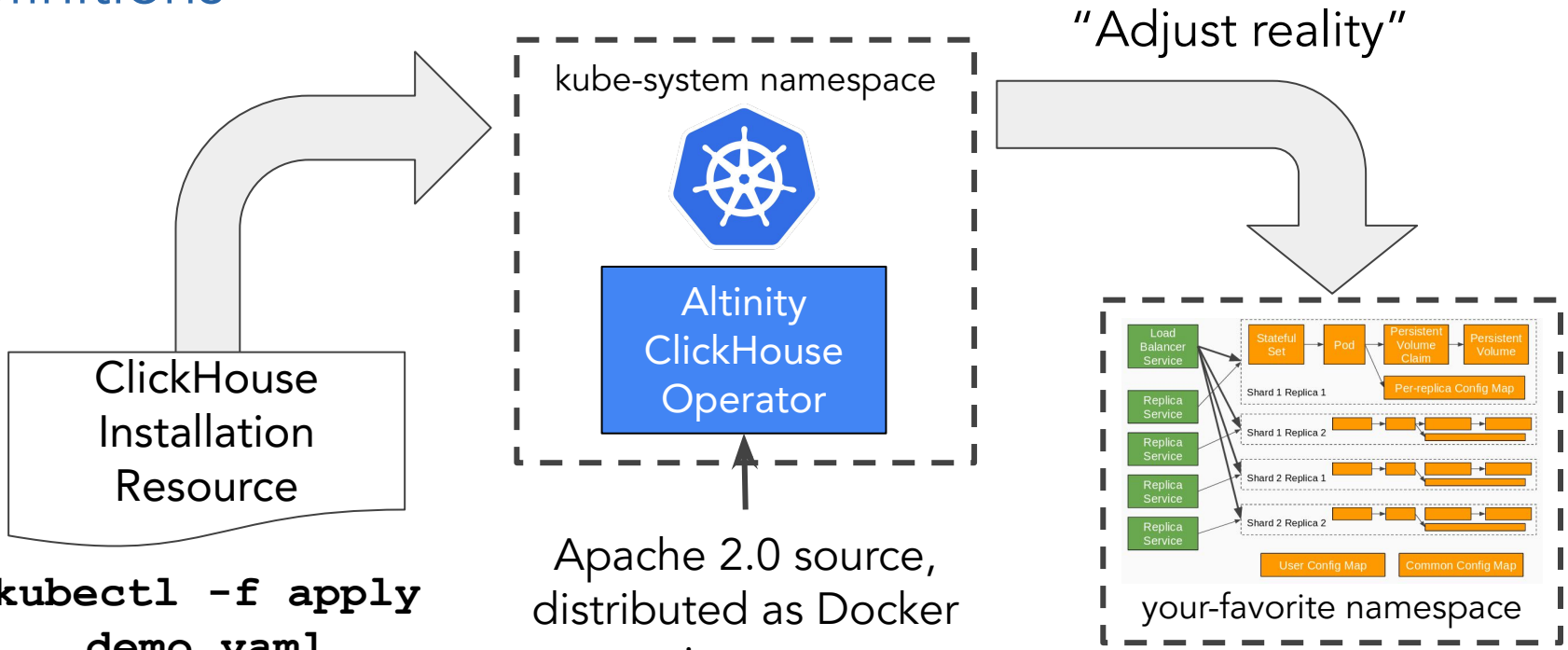
ClickHouse supports mTLS with authentication via client X509 certs

Kerberos

Can authenticate users

Managing ClickHouse users on Kubernetes

Altinity Operator sets up ClickHouse from CHI resource definitions



`kubectl -f apply demo.yaml`

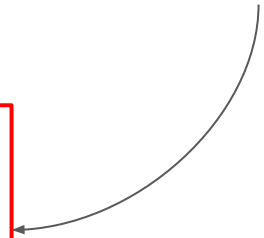
Apache 2.0 source, distributed as Docker image

Best practice deployment

Cluster resource definitions include users

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "demo"
spec:
  configuration:
    users:
      xmluser/networks/ip: "::/0"
      xmluser/password_sha256_hex: 5333...7a721
      xmluser/profile: "default"
      xmluser/access_management: 1
    zookeeper:
        nodes:
        - host: keeper-keeper
          port: 2181
```

Maps to XML users stored in /etc/clickhouse-server/users.d



You can transfer passwords using Kubernetes secrets

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"metadata:
  name: "demo"
spec:
  configuration:
    users:
      xmluser/networks/ip: "::/0"
      xmluser/password_sha256_hex:
        valueFrom:
          secretKeyRef:
            name: db-passwords
            key: xmlpass
      xmluser/profile: "default"
      xmluser/access_management: 1
    . . .
```

```
apiVersion: v1
kind: Secret
metadata:
  name: db-passwords
type: Opaque
data:
  xmlpass: NTgt...
```

CHI resources can also include raw XML files

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"metadata:
  name: "prod"
spec:
  configuration:
    profiles:
      readonly/readonly: "2"
    files:
      users.d/myquotas.xml:
        <clickhouse>
          <profiles>
            <default>
              <max_threads>4</max_threads>
              <log_queries>1</log_queries>
              . . .
```

Miscellaneous tricks for management

Introspection

System tables are your friends:

- `system.users`
- `system.roles, system.role_grants`
- `system.current_roles, system.enabled_roles`
- `system.settings_profiles`
- `system.settings where changed` – modified settings for the currently used profile
- `system.quotas`
- `system.quotas_usage`
- `system.row_policies`
- `system.user_directories`

And **`SHOW`** statement

Password Complexity Rules (config.xml)

```
<allow_plaintext_password>1</allow_plaintext_password>
<allow_no_password>1</allow_no_password>
<allow_implicit_no_password>1</allow_implicit_no_password>
<default_password_type>sha256_password</default_password_type>
```

Password Complexity Rules (disabled by default)

```
<password_complexity>
    <rule>
        <pattern>.{12}</pattern>
        <message>be at least 12 characters long</message>
    </rule>
    <rule>
        <pattern>\p{N}</pattern>
        <message>contain at least 1 numeric character</message>
    </rule>
    <rule>
        <pattern>\p{Ll}</pattern>
        <message>contain at least 1 lowercase character</message>
    </rule>
    <rule>
        <pattern>\p{Lu}</pattern>
        <message>contain at least 1 uppercase character</message>
    </rule>
    <rule>
        <pattern>[^\p{L}\p{N}]</pattern>
        <message>contain at least 1 special character</message>
    </rule>
</password_complexity>
```

Access Control Improvements (config.xml)

```
<access_control_improvements>
  <users_without_row_policies_can_read_rows>true</users_without_row_policies_can_read_rows>
  <on_cluster_queries_require_cluster_grant>true</on_cluster_queries_require_cluster_grant>
  <select_from_system_db_requires_grant>true</select_from_system_db_requires_grant>
  <select_from_information_schema_requires_grant>true</select_from_information_schema_requires_grant>
  <settings_constraints_replace_previous>true</settings_constraints_replace_previous>
  <table_engines_require_grant>false</table_engines_require_grant>
  <role_cache_expiration_time_seconds>600</role_cache_expiration_time_seconds>
</access_control_improvements>
```

Wrap-up

Takeaways on ClickHouse user management

- SQL RBAC offers a rich, DBA-friendly model that works over the network
- XML files are less feature rich but more developer-friendly
- Use [Zoo]Keeper replicated user directories in clusters
- Kubernetes deployments can use XML and/or RBAC
  - The Altinity Operator has extra features like support for secrets
- Not everything is documented!
  - Community lore and configuration files like config.xml are your friends

Something missing from ClickHouse user management?

Ask us.

We implemented existing features like LDAP support, Kerberos, and row policy templates. We can implement new features for you.

Email security bugs to security@altinity.com or security@clickhouse.com

More information!

- ClickHouse docs (https://clickhouse.com/docs)
- config.xml and users.xml – Lots of great comments
- Altinity Knowledge Base (https://kb.altinity.com/altinity-kb-setup-and-maintenance/rbac)
- Altinity blog (https://altinity.com/blog)
- Altinity Kubernetes Operator Hardening Guide
- Samples for this talk in GitHub

Thank you!
Questions?

Altinity.Cloud

Altinity Kubernetes Operator for ClickHouse

Altinity Stable Builds for ClickHouse

Website: https://altinity.com
Slack: https://www.altinity.com/slack

OSA Con 2024 - https://osacon.io