AWS Big Data Blog

# Secure data movement across Amazon S3 and Amazon Redshift using role chaining and ASSUMEROLE

by Sudipta Mitra, Lisa Matacotta, Michelle Deng, and Jared Cook | on 28 APR 2022 | in Amazon Redshift, Amazon SageMaker, Amazon Simple Storage Service (S3) | Permalink | Comments | Share

Data lakes use a ring of purpose-built data services around a central data lake. Data needs to move between these services and data stores easily and securely. The following are some examples of such services:

- Amazon Simple Storage Service (Amazon S3), which stores structured, unstructured, and semi-structured data
- Amazon Redshift, a fully managed, petabyte-scale data warehouse product to analyze large-scale structured and semi-structured data across data warehouses and operational databases
- Amazon SageMaker, which consumes data for machine learning (ML) capabilities

In multi-tenant architectures, groups or users within a group may require exclusive permissions to the group's S3 bucket and also the schema and tables belonging to Amazon Redshift. These teams also need to be able to control loading and unloading of data between the team-owned S3 buckets and Amazon Redshift schemas. Additionally, individual users within the team may require fine-grained control over objects in S3 buckets and specific schemas in Amazon Redshift. Implementing this permissions control use case should be scalable as more teams and users are onboarded and permission-separation requirements evolve.

Amazon Redshift and Amazon S3 provide a unified, natively integrated storage layer for data lakes. You can move data between Amazon Redshift and Amazon S3 using the Amazon Redshift COPY and UNLOAD commands.

This post presents an approach that you can apply at scale to achieve fine-grained access controls to resources in S3 buckets and Amazon Redshift schemas for tenants, including groups of users belonging to the same business unit down to the individual user level. This solution provides tenant isolation and data security. In this approach, we use the bridge model to store data and control access for each tenant at the individual schema level in the same Amazon Redshift database. We utilize ASSUMEROLE and role chaining to provide fine-grained access control when data is being copied and unloaded between Amazon Redshift and Amazon S3, so the data flows within each tenant's namespace. Role chaining also streamlines the new tenant onboarding process.

## Solution overview

In this post, we explore how to achieve resource isolation, data security, scaling to multiple tenants, and fine-grained access control at the individual user level for teams that access, store, and move data across storage using Amazon S3 and Amazon Redshift.

We use the bridge model to store data and control access for each tenant at the individual schema level in the same Amazon Redshift database. In the bridge model, a separate database schema is created for each tenant, and data for each tenant is stored in its own schema. The tenant has access only to its own schema.

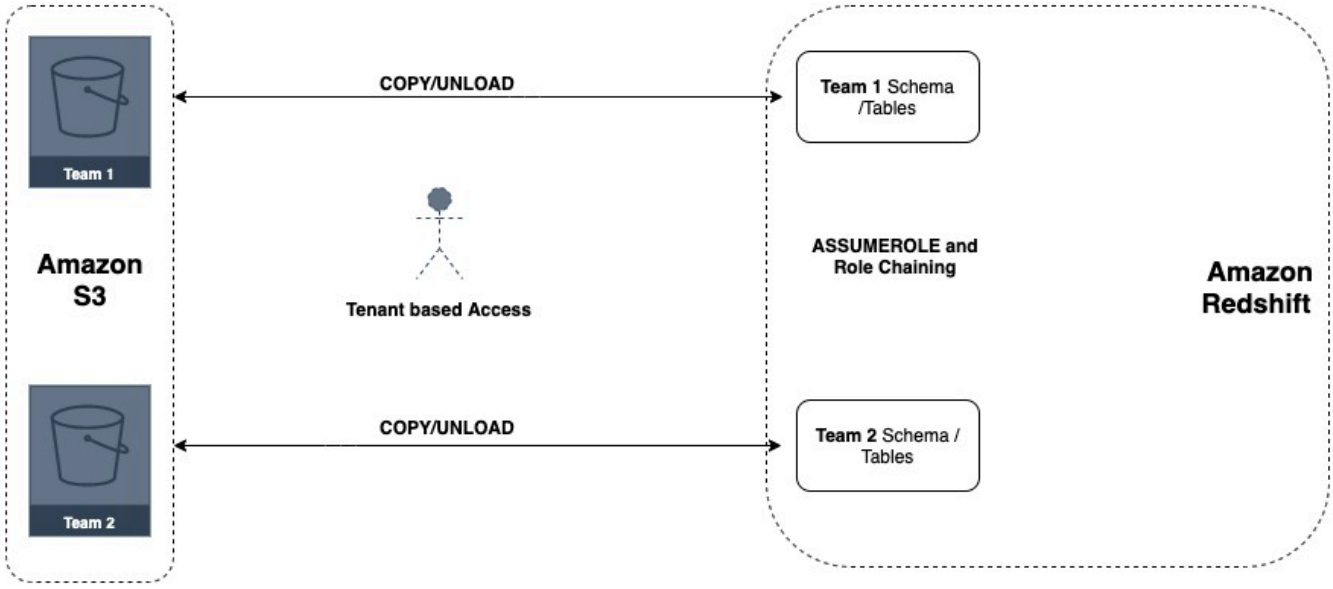
We use the COPY and UNLOAD commands to load and unload data into the Amazon Redshift cluster using an S3 bucket. These commands require Amazon Redshift to access Amazon S3 on your behalf, and security credentials are provided to your clusters.

We create an AWS Identity and Access Management (IAM) role—we call it the Amazon Redshift onboarding role—and associate it with the Amazon Redshift cluster. For each tenant, we create a tenant-specific IAM role—we call it the tenant role—to define the fine-grained access to its own Amazon S3 resources. The Amazon Redshift onboarding role doesn't have any permissions granted except allowing `sts:AssumeRole` to the tenant roles. The trust relationship to the Amazon Redshift onboarding role is defined in each of the tenant roles. We use the Amazon Redshift ASSUMEROLE privilege to control IAM role access privileges for database users and groups on COPY and UNLOAD commands.

Each tenant database user or group is granted ASSUMEROLE on the Amazon Redshift onboarding role and its own tenant role, which restricts the tenant to access its own Amazon S3 resources when using COPY and UNLOAD commands. We use role chaining when ASSUMEROLE is granted. This means that the tenant role isn't required to be attached to the Amazon Redshift cluster, and the only IAM role associated is the Amazon Redshift onboarding role. Role chaining streamlines the new tenant onboarding process. With role chaining, we don't need to modify the cluster; we can make modifications on the tenant IAM role definition when onboarding a new tenant.

For our use case, we have two tenants: team 1 and team 2. A tenant here represents a group of users—a team from the same business unit. We want separate S3 buckets and Amazon Redshift schemas for each team. These teams should be able to access their own data only and also be able to support fine-grained access control over copying and unloading data from Amazon S3 to Amazon Redshift (and vice versa) within the team boundary. We can apply access control at the individual user level using the same approach.

The following architecture diagram shows the AWS resources and process flow of our solution.




In this tutorial, you create two S3 buckets, two Amazon Redshift tenant schemas, two Amazon Redshift tenant groups, one Amazon Redshift onboarding role, and two tenant roles. Then you grant ASSUMEROLE on the

onboarding and tenant role to each tenant, using role chaining. To verify that each tenant can only access its own S3 resources, you create two Amazon Redshift users assigned to their own tenant group and run COPY and UNLOAD commands.

## Prerequisites

To follow along with this solution, you need the following prerequisites:

- An AWS account with permissions to access resources in Amazon Redshift, Amazon S3, AWS Key Management Service (AWS KMS), IAM, and AWS CloudFormation
- An Amazon Redshift cluster

## Download the source code to your local environment

To implement this solution in your local development environment, you can download the source code from the GitHub repo or clone the source code using the following command:

```
git clone https://github.com/aws-samples/amazon-redshift-assume-role-sample.git
```

The following files are included in the source code:

- **redshift-onboarding-role.cf.yaml** – A CloudFormation template to deploy the Amazon Redshift onboarding role `redshift-onboarding-role`
- **redshift-tenant-resources.cf.yaml** – A CloudFormation template to deploy an S3 bucket, KMS key, and IAM role for each tenant you want to onboard

## Provision an IAM role for Amazon Redshift and attach this role to the Amazon Redshift cluster

Deploy the template `redshift-onboarding-role.cf.yaml` using the AWS CloudFormation console or the AWS Command Line Interface (AWS CLI). For more information about stack creation, see Create the stack. This template doesn't have any required parameters. The stack provisions an IAM role named `redshift-s3-onboarding-role` for Amazon Redshift. The following code is the policy defining `sts:AssumeRole` to the tenant-specific IAM roles:

```
{
  "Version": "2012-10-17",
  "Statement": [
  {
   "Action": [
```

```
      "sts:AssumeRole"
     ],
    "Resource": [
      "arn:aws:iam::xxxxxxxxxxxx:role/*-tenant-redshift-s3-access-role"
     ],
    "Effect": "Allow"
    }
   ]
}
```

Navigate to the Amazon Redshift console and select the cluster you want to update. On the **Actions** menu, choose **Manage IAM roles**. Choose the role `redshift-s3-onboarding-role` to associate with the cluster. For more information, see Associate the IAM role with your cluster.

## Provision the IAM role and resources for tenants

Deploy the template `redshift-tenant-resources.cf.yaml` using the AWS CloudFormation console or the AWS CLI. For this post, you deploy the stack twice, supplying two unique tenant names for `TenantName`. For example, you can use `team1` and `team2` as the `TenantName` parameter values.

For each tenant, the stack provisions the following resources:

- A KMS key
- An S3 bucket named `team1-data-<account id>-<region>` with default encryption enabled with SSE-KMS using the created key
- An IAM role named `team1-tenant-redshift-s3-access-role`

The policy attached to the role `team1-tenant-redshift-s3-access-role` can only access the team's own S3 bucket. The role `redshift-s3-onboarding-role` is trusted to assume all tenant roles `*-tenant-redshift-s3-access-role` to enable role chaining. The tenant role `*-tenant-redshift-s3-access-role` has a trust relationship to `redshift-s3-onboarding-role`. See the following policy code:

```
        {
            "Action": [
                "s3:List*",
                "s3:Get*",
                "s3:Put*"
            ],
            "Resource": [
                "arn:aws:s3:::team1-data-<account id>-<region>/*",
                "arn:aws:s3:::team1-data-<account id>-<region>"
```

```
            ],
            "Effect": "Allow"
        }
```

## Create a tenant schema and tenant user with appropriate privileges

For this post, you create the following Amazon Redshift database objects using the query editor on the Amazon Redshift console or a SQL client tool like SQL Workbench/J. Replace <password> with your password and <account id> with your AWS account ID before running the following SQL statements:

```
create schema team1;
create schema team2;

create group team1_grp;
create group team2_grp;

create user team1_usr with password '<password>' in group team1_grp;
create user team2_usr with password '<password>' in group team2_grp;

grant usage on schema team1 to group team1_grp;
grant usage on schema team2 to group team2_grp;

GRANT ALL ON SCHEMA team1 TO group team1_grp;
GRANT ALL ON SCHEMA team2 TO group team2_grp;

revoke assumerole on all from public for all;

grant assumerole
```

## Verify that each tenant can only access its own resources

To verify your access control settings, you can create a test table in each tenant schema and upload a file to the tenant's S3 bucket using the following commands. You can use the Amazon Redshift query editor or a SQL client tool.

1. Sign in as `team1_usr` and enter the following commands:

```
CREATE TABLE TEAM1.TEAM1_VENUE(
VENUEID SMALLINT,
VENUENAME VARCHAR(100),
VENUECITY VARCHAR(30),
VENUESTATE CHAR(2),
```

```
VENUESEATS INTEGER
) DISTSTYLE EVEN;

commit;
```

2. Sign in as `team2_usr` and enter the following commands:

```
CREATE TABLE TEAM2.TEAM2_VENUE(
VENUEID SMALLINT,
VENUENAME VARCHAR(100),
VENUECITY VARCHAR(30),
VENUESTATE CHAR(2),
VENUESEATS INTEGER
) DISTSTYLE EVEN;

commit;
```

3. Create a file named `test-venue.txt` with the following contents:

```
7|BMO Field|Toronto|ON|0
16|TD Garden|Boston|MA|0
23|The Palace of Auburn Hills|Auburn Hills|MI|0
28|American Airlines Arena|Miami|FL|0
37|Staples Center|Los Angeles|CA|0
42|FedExForum|Memphis|TN|0
52|PNC Arena|Raleigh|NC|0
59|Scotiabank Saddledome|Calgary|AB|0
66|SAP Center|San Jose|CA|0
73|Heinz Field|Pittsburgh|PA|65050
```

4. Upload this file to both `team1` and `team2` S3 buckets.

5. Sign in as `team1_usr` and enter the following commands to test Amazon Redshift COPY and UNLOAD:

```
copy team1.team1_venue
from 's3://team1-data-<account id>-<region>/'
iam_role 'arn:aws:iam::<account id>:role/redshift-s3-onboarding-role,arn:aws:iam::<a
delimiter '|' ;

unload ('select * from team1.team1_venue')
```

```
to 's3://team1-data-<account id>-<region>/unload/'
iam_role 'arn:aws:iam::<account id>:role/redshift-s3-onboarding-role,arn:aws:iam::<a
```

The file test-venue.txt uploaded to the `team1` bucket is copied to the table `team1_venue` in the team1 schema, and the data in table `team1_venue` is unloaded to the `team1` bucket successfully.

6. Replace `team1` with `team2` in the preceding commands and then run them again, this time signed in as `team2_usr`.

If you're signed in as `team1_usr` and try to access the `team2` S3 bucket or `team2` schema or table and team2 IAM team1 resources while logged in as `team2_usr`.

## Clean up

To clean up the resources you created, delete the CloudFormation stack created in your AWS account.

## Conclusion

In this post, we presented a solution to achieve role-based secure data movement between Amazon S3 and Amazon Redshift. This approach combines with the ASSUMEROLE feature in Amazon Redshift to allow fine-grained access control over the COPY and UNLOAD commands down to the individual user level within a particular team. This in turn provides finer control over resource isolation and data security in a multi-tenant solution. Many use cases can benefit from this solution as more enterprises build data platforms to provide the foundations for highly scalable, customizable, and secure data consumption models.

---

### About the Authors

**Sudipta Mitra** is a Senior Data Architect for AWS, and passionate about helping customers to build modern data analytics applications by making innovative use of latest AWS services and their constantly evolving features. A pragmatic architect who works backwards from customer needs, making them comfortable with the proposed solution, helping achieve tangible business outcomes. His main areas of work are Data Mesh, Data Lake, Knowledge Graph, Data Security and Data Governance.

**Michelle Deng** is a Sr. Data Architect at Amazon Web Services. She works with AWS customers to provide guidance and technical assistance about database migrations and Big data projects.

**Jared Cook** is a Sr. Cloud Infrastructure Architect at Amazon Web Services. He is committed to driving business outcomes in the cloud, and uses Infrastructure as Code and DevOps best practices to build resilient architectures on AWS.  In his leisure time, Jared enjoys the outdoors, music, and plays the drums.

**Lisa Matacotta** is a Senior Customer Solutions Manager at Amazon Web Services. She works with AWS customers to help customers achieve business and strategic goals, understand their biggest challenges and provide guidance based on AWS best practices to overcome them.

## Comments