

Lake Formation: Data Security and Data Governance with LF-TBAC

12 November 2021 - 10 min. read

AWS Glue

AWS Identity and Access Management (IAM)

AWS Lake Formation

Data Security and Governance

Database

Introduction

Big Data has rapidly grown as a way to describe information obtained from heterogeneous sources when it becomes incredibly complex to manage in terms of **Variety, Veracity, Value, Volume, and Velocity**. Still, it can be considered the “New Gold because of the potential to generate business value.”

Without adequate governance or quality, data lakes can quickly turn into unmanageable data swamps. Data engineers know the data they need lives in these swamps, but they won't be able to find, trust, or use it without a clear data governance strategy.

A very common challenge is **maintaining** Governance, **access control** over users who operate on the Data Lake, and protecting sensitive information.

Companies need to centralize governance, access control, and a strategy backed by managed services to fine-grain control user access to data.

Dealing with these situations typically requires two approaches: *manual*, **more flexible** but **complex**; *managed* which **requires your solution to fit into specific standards** but in return **takes away all management complexities** for the developers.

This article will guide you through setting up your Data Lake with Lake Formation, showing all the challenges that must be addressed during the process with a particular eye on Security and Governance through the LF-TBAC approach.

Tag-Based Access Control, in short **TBAC**, is an increasingly popular way to solve these challenges, applying constraints based on tags associated with specific resources.

So, without further ado, let's dig in!

What is TBAC access

Tag-based access control allows administrators of IAM-enabled resources to create access policies based on existing tags associated with eligible resources.

Cloud providers manage permissions of both users and applications with policies, documents with rules that reference resources. By applying tags to those resources is possible to define simple and effective allow/deny conditions.

Using access management tags may reduce the number of access policies needed within a cloud account while also providing a simplified way to grant access to a heterogeneous group of resources.

Why S3 alone is not enough

S3, like most AWS services, **leverages the IAM principals for access management**, meaning that it is possible to define which parts of a bucket (files and folders/prefixes) a single IAM principal can read/write; however is not possible to further restrict IAM access to specific parts of an object, nor to certain data segments stored inside objects.

For example, let's assume that our application data is stored as a collection of parquet files divided per country in different folders.

It is possible to constrain a user to access only the users belonging to a given country. Still, there is no way to prevent them from reading the anagraphic information (e.g., username and address) stored as columns in the parquet.

The **only way to prevent users from accessing sensitive information would be to encrypt the columns before writing the files to S3**, which can be **slow, cumbersome**, and open a whole new 'can of worm' regarding **key storage, sharing**, and eventually **key decommissioning**.

Furthermore, **giving access to external entities using IAM principals is often a non-trivial problem on its own.**

Luckily, AWS offers a **battery included solution to the S3 Data Lake permission problem**: enters AWS Lake Formation!

AWS Lake Formation is a fully managed service that simplifies building, securing, and managing data lakes, automating many of the complex manual steps required to create them.

Lake Formation also provides **its own permissions model, which is what we want to explore in detail, that augments the classical AWS IAM permissions model.**

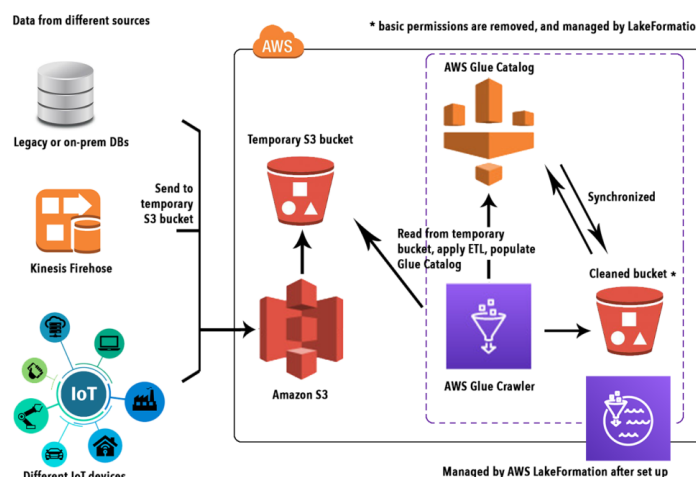
This centrally defined permissions model enables fine-grained access to data stored in data lakes through a simple grant/revoke mechanism.

So, by leveraging the power of Lake Formation, we would like to demonstrate, with a simple solution, how to address the aforementioned S3 challenges; let's continue!

Leveraging TBAC approach in Lake Formation

To accompany the reader in understanding why AWS Lake Formation can be a good choice in dealing with the complexities of managing a DataLake, we have prepared a simple tutorial on how to migrate heterogeneous data.

From legacy on-prem databases into S3 while also creating a Lake Formation catalog to deal with data cleansing, permissions, and further operations.



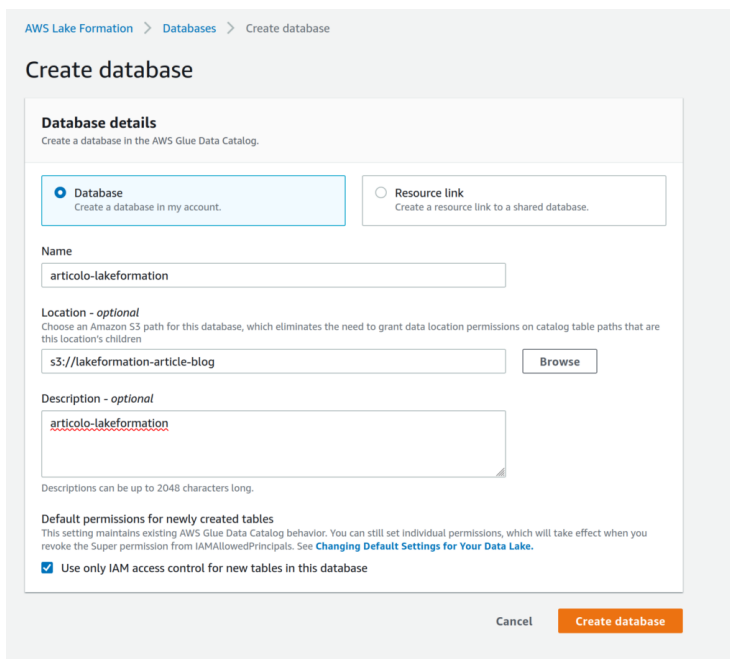
Our example implementation

AWS Glue migration of on-prem data

The first step for creating a Data Lake is obviously to fetch, transform and insert the data. In this simple example, we used a mocked users dataset from a MySQL database. AWS glue is the natural way to connect to the heterogeneous data source, infer their schema import and transform the data and finally write them on S3 [as we explained in detail here](#).

After the data is loaded in a temporary S3 bucket, you need to create a **Database in Lake Formation** to connect to a **Glue Crawler** and run it on your S3 prefix to populate a Glue Catalog for your data.

Just go to the **AWS Lake Formation console**, in the *Databases* page under the **Data catalog tab**, and fill in a Database name and your S3 path.



Create a new Database from Lake Formation

Note: creating a database from Lake Formation assures correct permissions are associated with it, we could have done the same thing from AWS Glue but we would have needed extra effort to modify permissions for the next steps.

After the database is created, we need the Glue Catalog, which is a metastore containing the schema (schema-on-read) of your data saved in S3 (usually as parquet files).

Having a Glue Schema is **necessary to set up the AWS Lake Formation access layer in**

front of your S3 Data Lake. To make it, just create a Crawler and link it to the same S3 path as the Database, and **set that DB as the crawler output.**

The screenshot shows the 'Add crawler' wizard in AWS Glue, specifically the 'Add a data store' step. On the left, a sidebar lists the steps: 'Crawler info', 'Crawler source type', 'Data stores', 'IAM Role', 'Schedule', 'Output', and 'Review all steps'. The 'Data stores' step is currently selected. The main area is titled 'Add a data store'. It has a dropdown for 'Choose a data store' set to 'S3'. Below that is a 'Connection' dropdown set to 'Select a connection'. There is an 'Add connection' button. The 'Crawl data in' section has two radio buttons: 'Specified path in my account' (selected) and 'Specified path in another account'. The 'Include path' field contains 's3://lakeformation-article-blog'. Below this is a note: 'All folders and files contained in the include path are crawled. For example, type s3://MyBucket/MyFolder/ to crawl all objects in MyFolder within MyBucket.' The 'Sample size (optional)' field has a note: 'Enter an integer between 1 and 249.' Below that is a note: 'This field sets the number of files in each leaf folder to be crawled. If not set, all the files are crawled.' There is an 'Exclude patterns (optional)' section. At the bottom are 'Back' and 'Next' buttons.

Setup of a basic AWS Glue Crawler

In order to use the Crawler, an IAM role is necessary, but luckily AWS has a step for that in the Crawler creation wizard:

The screenshot shows the 'Add crawler' wizard in AWS Glue, specifically the 'Choose an IAM role' step. It has three radio buttons: 'Update a policy in an IAM role', 'Choose an existing IAM role', and 'Create an IAM role' (selected). Below the radio buttons is the 'IAM role' section. It shows 'AWSGlueServiceRole-' followed by a text input field containing 'lake-formation-article'. Below this is a list of permissions: 's3://lakeformation-article-blog'. At the bottom are 'Back' and 'Next' buttons.

How to create an IAM role for using the Crawler

Once the Crawler is created, and data is imported into the catalog, we are ready for the next step.

The screenshot shows the AWS CloudWatch Logs console. The breadcrumb trail is 'CloudWatch > Log groups > /aws-glue/crawlers > article-lakeformation-crawler'. The 'Log events' section is active. A search filter is applied: '646ba47-4512-4d7b-ab13-d76f66d7561'. The log events are as follows:

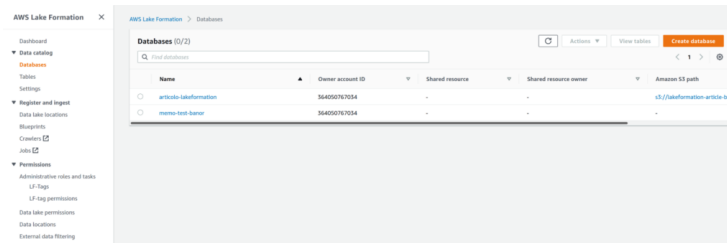
| Timestamp | Message |
|-------------------------------|---|
| 2021-10-18T16:53:39.742+02:00 | [646ba47-4512-4d7b-ab13-d76f66d7561] BENCHMARK : Running Start Crawl for Crawler article-lakeformation-crawler |
| 2021-10-18T16:53:57.818+02:00 | [646ba47-4512-4d7b-ab13-d76f66d7561] BENCHMARK : Classification complete, writing results to database article-lakeformation |
| 2021-10-18T16:53:57.812+02:00 | [646ba47-4512-4d7b-ab13-d76f66d7561] INFO : Crawler configured with SchedulePolicy {"UpdateBehavior":"UPDATE_IN_DATABASE","DeleteBehavior":"DELETE_...} |
| 2021-10-18T16:54:19.821+02:00 | [646ba47-4512-4d7b-ab13-d76f66d7561] INFO : Created table lakeformation_article_blog in database article-lakeformation |
| 2021-10-18T16:54:21.768+02:00 | [646ba47-4512-4d7b-ab13-d76f66d7561] BENCHMARK : Finished writing to catalog |
| 2021-10-18T16:55:28.892+02:00 | [646ba47-4512-4d7b-ab13-d76f66d7561] BENCHMARK : Crawler has finished running and is in state READY |

Cloudwatch Logs demonstrating that Crawler worked correctly

AWS Lake Formation

By having a Glue Data catalog in place, it is time to set up Lake Formation to finally manage user access permissions.

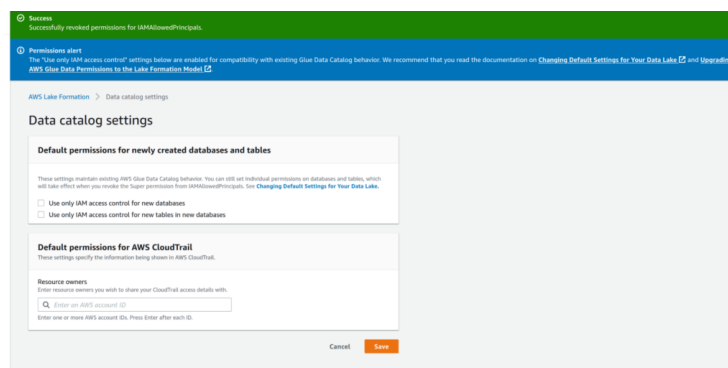
In order to do so, let's start by going to the Lake Formation dashboard and **removing the usual S3 access permissions**.



Lake Formation dashboard

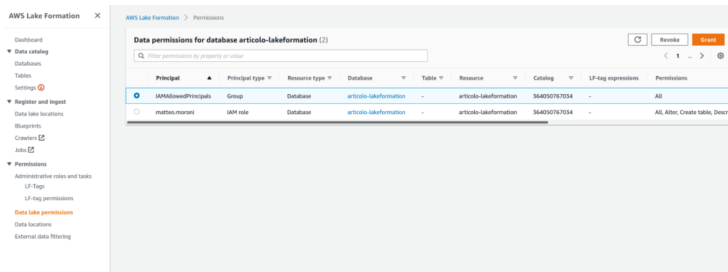
So we can go to *Data Catalog Settings* and uncheck *Use only IAM access control for new databases* and *Use only IAM access control for new tables in new databases*.

By default, access to Data Catalog resources and Amazon S3 locations are controlled solely by AWS Identity and Access Management (IAM) policies, unchecking the values allows Individual Lake Formation **permissions** to take effect.



Lake Formation data catalog setting: disable both the Use only flag

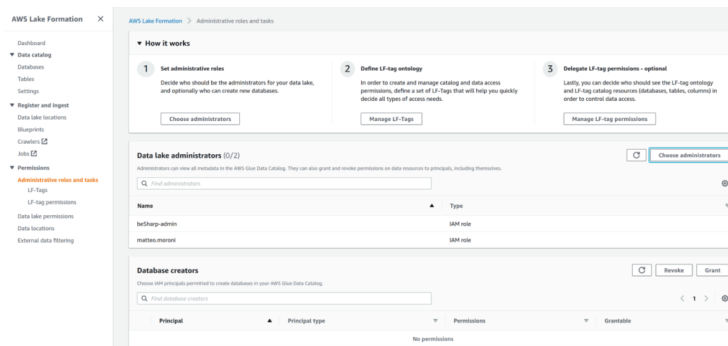
Once access **responsibilities are delegated to Lake Formation**, we can remove the access for the standard IAMAllowedPrincipals IAM group, in the data lake *Permissions* tab, select the **permission of the IAM group** and click *Revoke*.



Revoke standard IAMAllowedPrincipals permissions

The user creating the DataLake will also be listed in this section with admin privileges, if you want that user to retain access to the data you can leave the permission as they are, otherwise you can either **revoke the permission to the user/role or restrict them**.

*Note: if you need to add a Data lake administrator principal, you can do so by going to the Administrative roles and tasks and adding a **Data lake admin**.*



Add admin and db creator console

Once all these steps are completed, it is time to start defining Lake Formation tags (**LF-Tags** from now on), which will be used to restrict access to the data lake.

From the *LF-Tags* page under the *Permissions* tab **create a new LF-Tag** and for key use *level* and add *private*, *sensitive*, and *public* as value separated by comma just like in the figure. Click **Add LF-tag**.

Add LF-Tag [Learn More](#) ✕

LF-Tags have a key and one or more values that can be associated with data catalog resources. Tables automatically inherit from database LF-tags, and columns inherit from table LF-tags.
Example: Key = Confidentiality | Values = private, sensitive, public

Key

Key string must be less than 128 characters long, and cannot be changed once LF-tag is created.

Values

Type a single value and select [Enter] or specify multiple values separated by commas.

Enter up to 15 values; each value must be less than 256 characters long.

LF-Tag creation

Now once created, how can we use these tags to enforce access control? First of all, let's go to the database section and **select our database**, created at the beginning of the tutorial. In *database actions*, you can select the tag you've created and the permission level.

Usually, we leave the database access open and restrict permissions on a per table and fields basis, but this is different for each database. In our example, we assign the level **public** to the whole example database.

Edit LF-Tags: articolo-lakeformation [Learn More](#) ✕

LF-Tags

After they are associated with catalog resources, LF-Tags allow you to create scalable permissions.

Assigned keys

✕

Values

▼

You can add 49 more LF-tags.

Edit LF-Tag for the entire database

Now if we want to **restrict access to the columns in the user table containing personal info**, we can go to the table to modify, select the column and change its LF-tag from **public** to **private** (see figures).

lakeformation_article_blog Version 0 (Current version) ▾

Schema Upload Schema Delete Edit Edit tags Add column

Find Columns < 1 2 > ⚙

| | # | Column Name ▾ | Data ty... ▾ | Partition key | Comment |
|-------------------------------------|----|--------------------------------|--------------|---------------|---------|
| <input type="checkbox"/> | 1 | id | string | - | - |
| <input checked="" type="checkbox"/> | 2 | first_name | string | - | - |
| <input type="checkbox"/> | 3 | middle_name | string | - | - |
| <input type="checkbox"/> | 4 | last_name | string | - | - |
| <input type="checkbox"/> | 5 | email | string | - | - |
| <input type="checkbox"/> | 6 | username | string | - | - |
| <input type="checkbox"/> | 7 | password | string | - | - |
| <input type="checkbox"/> | 8 | sex | string | - | - |
| <input type="checkbox"/> | 9 | telephone_number | string | - | - |
| <input type="checkbox"/> | 10 | date_of_birth | string | - | - |
| <input type="checkbox"/> | 11 | age | double | - | - |
| <input type="checkbox"/> | 12 | company_email | string | - | - |
| <input type="checkbox"/> | 13 | national_identity_card_number | string | - | - |
| <input type="checkbox"/> | 14 | national_identification_num... | double | - | - |
| <input type="checkbox"/> | 15 | passport_number | string | - | - |
| <input type="checkbox"/> | 16 | bank_account | string | - | - |

Schema of our example database in which we select a column

Edit LF-Tags: first_name [Learn More](#) [🔗](#) ✕

LF-Tags
After they are associated with catalog resources, LF-Tags allow you to create scalable permissions.

Inherited keys Values

You can add 49 more LF-tags.

Editing a per column LF-Tag

Now we just need to define which IAM principals (i.e, our test user) will have access to a given LF-Tag. To do so, let's go to *Data lake permissions* and **grant permissions to an IAM user/role/group to access resources tagged with a given LF-Tag**.

Grant read permission

This example shows how to give a user access to all the resources tagged with “level”: “public”.

This user will thus be able to see all our databases except for the personal data tagged as private. Another user may have access to both public and private information, just add the private level in the LF-Tag section or modify columns tags according to your needs.

We can now query the database table using our test user which, based on our set of permissions, is not able to see the first_name column (which is tagged as private).

Athena is used to querying data and demonstrating that first_name is not shown in the table select because is tagged as private

As shown in the figure we have successfully managed to deny our test user the right to see a “sensible” column of our choice.

We would like to encourage the user to experiment in adding or removing also describe and select options from the LF-Tag permissions in the Data Lake section to

see that we can also deny listing both database and tables.

Note: as of Nov 3, 2021: to enhance security, AWS Lake Formation also added support for managed VPC endpoints via [AWS PrivateLink](#) to access a data lake in a Virtual Private Cloud.

Feature in preview: row-level security

Lake Formation is still a young service, so there is much room for improvement. AWS is constantly working on increasing features for its services, and Lake Formation is no exception.

AWS Lake Formation already allows setting access policies to hide data, such as a column with sensitive information, from users who do not have permission to view that data.

Row-level security will add up to that by allowing to set row-level policies in addition to column-level policies.

An example could be setting a policy that gives a data scientist access to only the experiment data marked with a specific id.

Another interesting aspect would be to share the same Data Lake for different datasets to reduce costs and management efforts.

To Sum up

In this article, we have seen how we can leverage the power of AWS Services for Storage and Data Analytics to tackle the challenge imposed by Big Data, in particular how to manage access, permissions, and governance.

We have shown that AWS Glue crawlers can effectively retrieve unstructured data from temporary repositories, being them databases like RDS or on-premises, or object storages like S3, and obtain a schema to populate a Glue Catalog.

We have seen that starting from S3 and a metadata store, it is possible to create a Lake Formation Catalog on top of S3, entirely managed by AWS, to drastically reduce the management effort to set up and administrate a Data lake.

We have briefly seen what is a Tag-Based Access Control (TBAC) methodology and how can be effectively used to manage access and permissions.

We have shown that AWS Lake Formation can apply IAM policies and TBAC rules to give or restrain access to users and groups even on a per-column/row basis. We demonstrated that with Lake Formation and AWS Glue, we could obscure sensitive data to specific principals.

We have described LF-Tags in detail, with a simple tutorial. Finally, We have talked about Row-Level Security.

To conclude, we can say that for challenges regarding Big Data and proper storage solutions, with an eye for security and governance matters, there are always two possible choices to make: DIY or opt for a managed solution.

In this article, we chose a **managed** solution to show all the benefits of a more rigid approach to the problem. Despite being less flexible to adaptation, it offers a service more adherent to best practices and less burden in administration and governance.

As always, feel free to comment in the section below, and reach us for any doubt, question or idea!

See you on **Proud2beCloud** in a couple of weeks for a new story!



Alessandro Gaggia

Head of software development at beSharp and Full-Stack Developer, I keep all our codebases up-to-date. I write code in almost any language, but Typescript is my favorite. I live for IT, Game design, Cinema, Comics, and... good food. Drawing is my passion!



Matteo Moroni

DevOps and Solution Architect at beSharp, I deal with developing SaaS, Data Analysis, and HPC solutions, and with the design of unconventional architectures with different

complexity. Passionate about computer science and physics, I have always worked in the first and I have a PhD in the second. Talking about anything technical and nerdy makes me happy!

Copyright © 2011-2021 by beSharp srl - P.IVA IT02415160189