

Article

[Anton Umnikov](#) · Oct 17, 2019 5m read

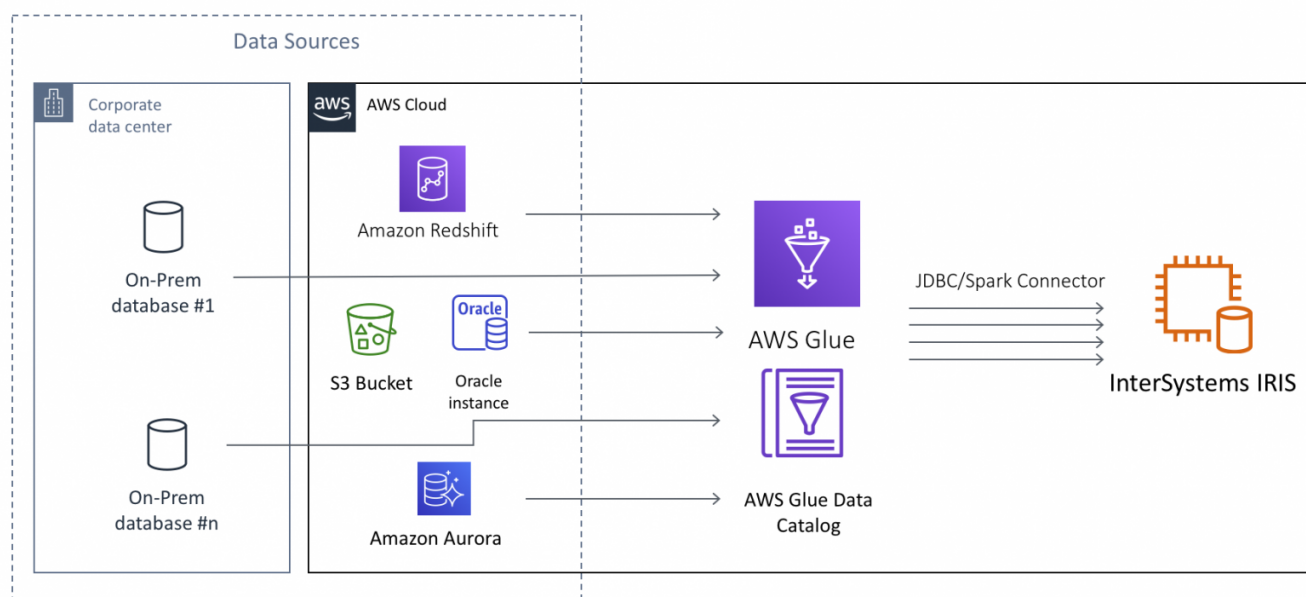
Using AWS Glue with InterSystems IRIS

October 17, 2019

Anton Umnikov
Sr. Cloud Solutions Architect at InterSystems
AWS CSAA, GCP CACE

[AWS Glue](#) is a fully managed ETL (extract, transform, and load) service that makes it simple and cost-effective to categorize your data, clean it, enrich it, and move it reliably between various data stores.

In the case of InterSystems IRIS, AWS Glue allows moving large amounts of data from both Cloud and on-Prem data sources into IRIS. Potential data sources include, but not limited to on-Prem databases, CSV, JSON, Parquet and Avro files residing in S3 buckets, Cloud-native databases such as AWS Redshift and Aurora and many others.



This article assumes that you have the basic familiarity with AWS Glue, at least at the level of completing [AWS Glue Getting Started tutorials](#). We will concentrate on the technical aspects of configuring AWS Glue Jobs to use InterSystems IRIS as a Data Target, or in other terms - "data sink".

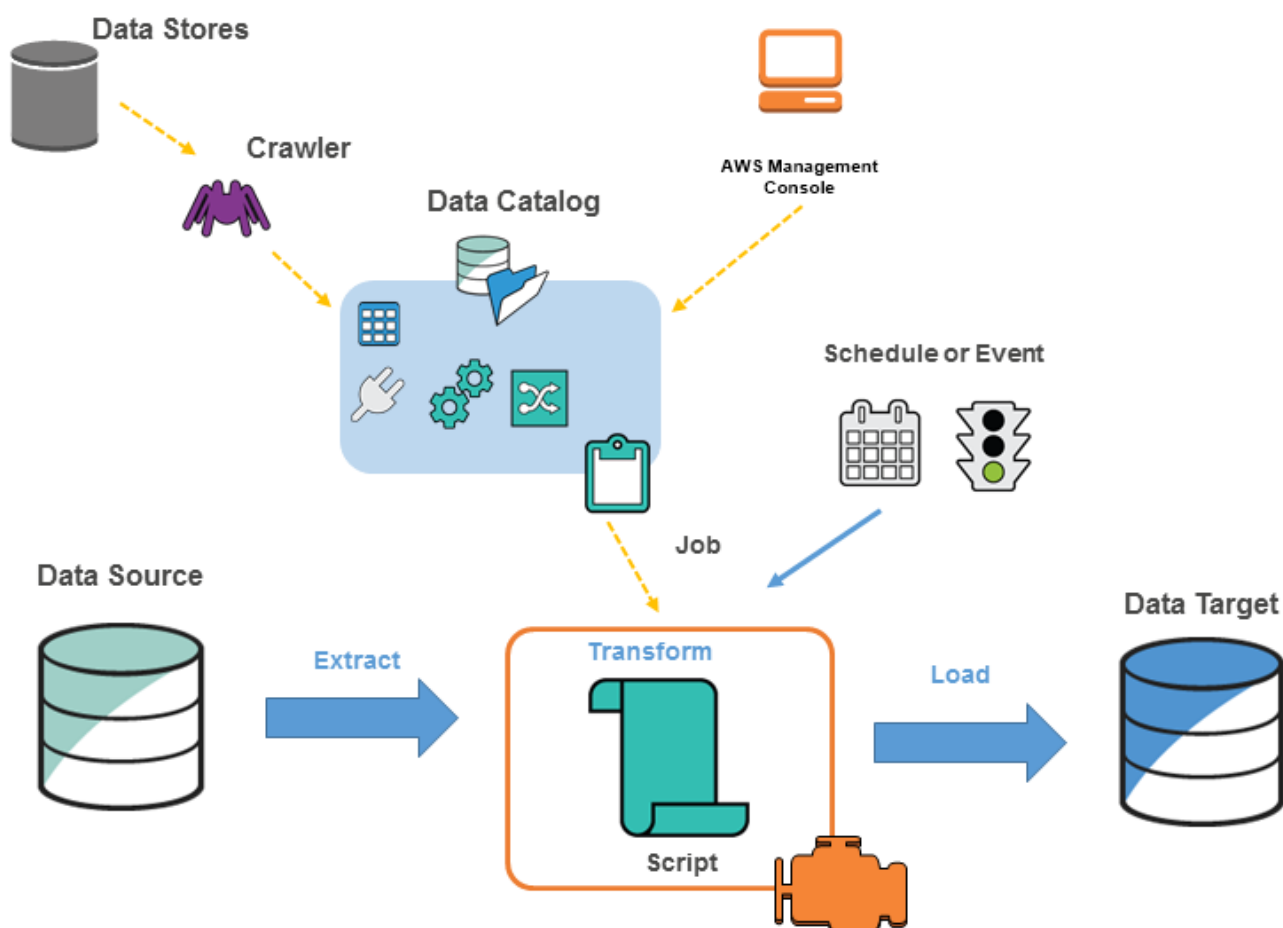
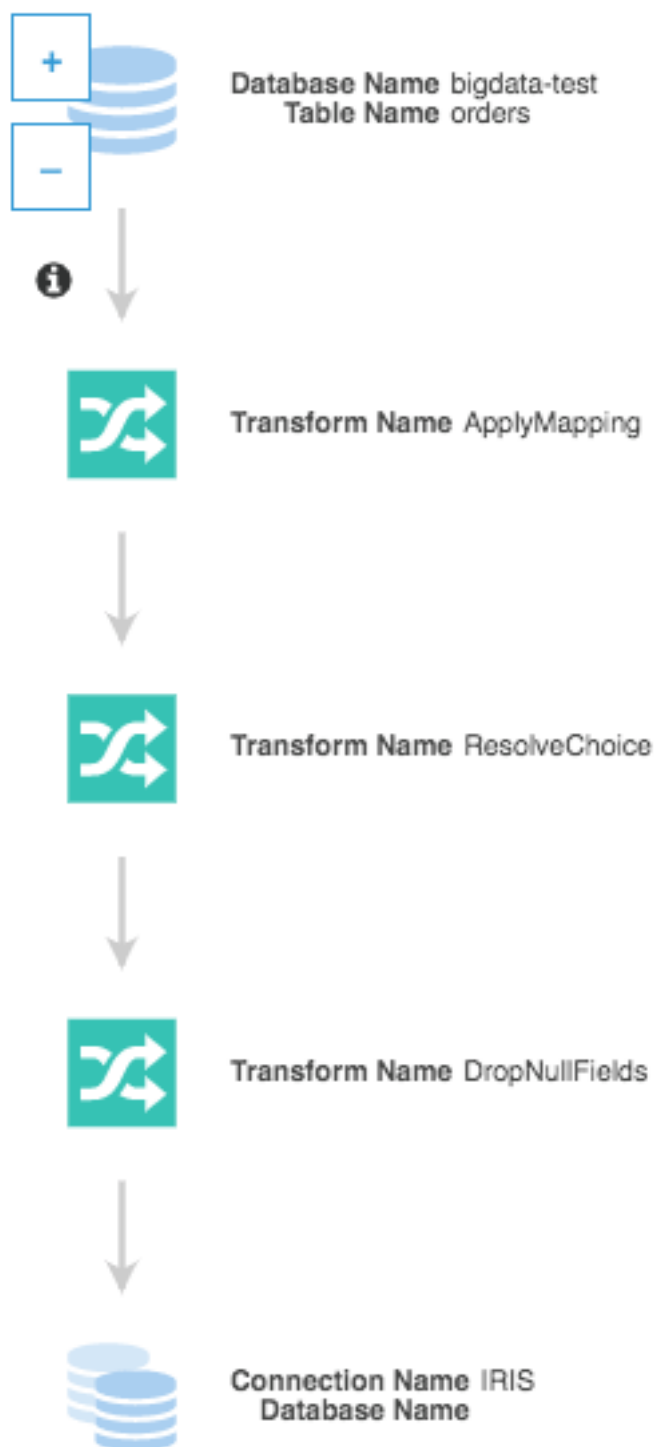


Image from <https://docs.aws.amazon.com/glue/latest/dg/components-key-concepts.html>

AWS Glue Jobs are run in a "Serverless" manner. All the resources, required for performing the job are dynamically provisioned by AWS only for the time the job is actually running and immediately destroyed the moment the job is completed, so instead of provisioning, managing and incurring ongoing costs for the required infrastructure you are being billed only for the time job is actually running and spend your efforts only on writing the Job code. At the "idle" time - no resources are consumed, other than S3 buckets, storing the job code and configuration.



While multiple options are available, typically Glue Jobs are executed on dynamically provisioned Apache Spark and written in PySpark code. Glue Job contains the "Extract" portion, where data is being extracted from data sources, series of "Transformations", build using Glue API and finally the "Load" or "sink" part, where after final transformation data is being written to the target system.

In order to enable AWS Glue to interact with IRIS we need to ensure the following:

- Glue has network access to the IRIS instances involved
- IRIS JDBC driver JAR file is accessible to the Glue Job
- Glue Job is using API, compatible with InterSystems IRIS JDBC

Let's examine each of the required steps.

Create IRIS Connection

In AWS Console select AWS Glue->Connections->Add Connection

AWS Glue

Connections A connection contains the properties needed to connect to your data.

[Add connection](#) [Test connection](#) [Action](#) Showing: 1 - 1

<input type="checkbox"/>	Name	Type	Date created	Last updated	Updated by
<input type="checkbox"/>	IRIS	JDBC	16 October 2019 ...	16 October 2019 ...	user/antonum

[Feedback](#) [English \(US\)](#) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy Policy](#) [Terms of Use](#)

Enter the name of your connection and select "JDBC" for Connection Type.

Set up your connection's properties.

For more information, see [Working with Connections](#).

Connection name

Connection type

Choose database engine...

Amazon Redshift

Amazon RDS

JDBC

JDBC

Enter description...

Next

In JDBC URL enter JDBC connection string for your IRIS instance, username and password information.

The next step is crucial - you need to make sure that Glue places its endpoints into the same VPC as your IRIS instance. Select VPC and Subnet of your IRIS instance. Any security group with a self-referencing inbound rule for all TCP ports would do here. For instance - your IRIS Instance security group.

IAM Role with access to JDBC driver

If you haven't done it already - upload IRIS JDBC driver JAR file `intersystems-jdbc-3.0.0.jar` into S3 bucket. In this example, I'm using `s3://irisdistr` bucket. It would be different for your account.




You need to create IAM Role for your Glue Job, that can access that file, along with other S3 buckets that Glue would use to store scripts, logs etc.

Make sure it has the read access to the bucket of your JDBC driver. In this case, we granting this role (GlueJobRole) `ReadOnly` access to all buckets, along with predefined `AWSGlueServiceRole`. You might choose to further limit this role permissions.

[Roles](#) > [GlueJobRole](#)

Summary




[Delete role](#)

Role ARN	arn:aws:iam::  role/GlueJobRole 
Role description	Allows Glue to call AWS services on your behalf. Edit
Instance Profile ARNs	
Path	/
Creation time	2019-10-16 21:57 EDT
Maximum CLI/API session duration	1 hour Edit

[Permissions](#)[Trust relationships](#)[Tags](#)[Access Advisor](#)[Revoke sessions](#)

▼ Permissions policies (3 policies applied)

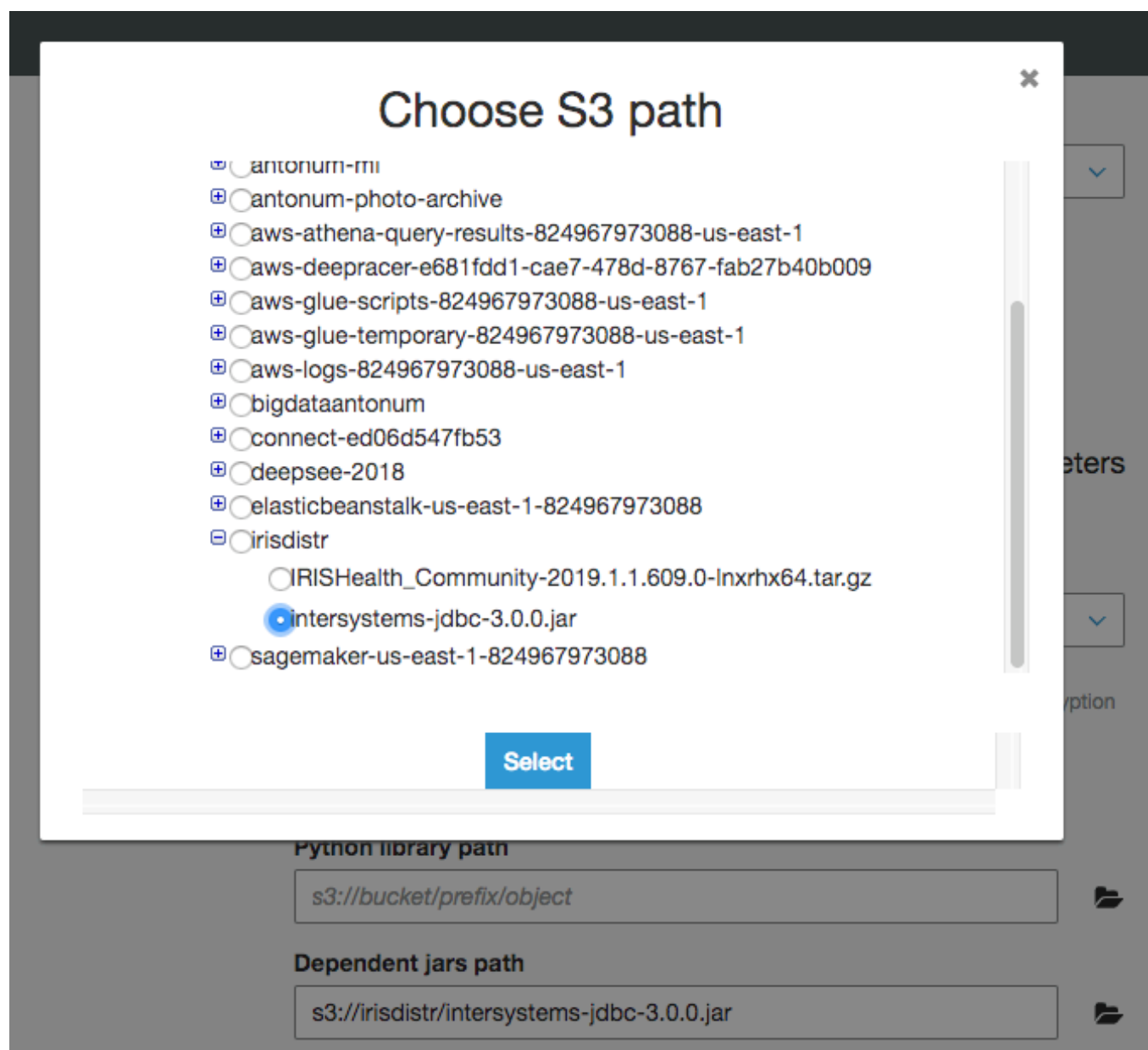
[Attach policies](#)[+ Add inline policy](#)

	Policy name ▼	Policy type ▼	
▶	 SecretsManagerReadWrite	AWS managed policy	✕
▶	 AWSGlueServiceRole	AWS managed policy	✕
▶	 AmazonS3ReadOnlyAccess	AWS managed policy	✕

Create and configure Glue Job

Create a new Glue Job. Select IAM Role, we created in the previous step. Leave everything else default.

Under "Security configuration, script libraries, and job parameters (optional)" set "Dependent jars path" to the location of the `intersystems-jdbc-3.0.0.jar` in the S3 bucket.



For Source - use one of your pre-existing data sources. If you followed tutorials, referenced above you'll have at least one already.

Use "Create tables in your data target" option and select IRIS connection you've created in the previous step. Leave everything else default.

If you followed us up until now you should arrive at the screen similar to this:

Job: IRIS-DataLoad

Database Name sampledb
Table Name elb_logs

Transform Name ApplyMapping

Transform Name ResolveChoice

Transform Name DropNullFields

```

18 ## @return: datasource0
19 ## @inputs: []
20 datasource0 = glueContext.create_dynamic_frame.from_catalog(database = "sampledb", table_name = "elb_logs",
21 ## @type: ApplyMapping
22 ## @args: [mapping = [{"request_timestamp": "string", "request_timestamp": "string"}, {"elb_name": "string"}],
23 ## @return: applymapping1
24 ## @inputs: [frame = datasource0]
25 applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [{"request_timestamp": "string", "request_timestamp": "string"}, {"elb_name": "string"}],
26 ## @type: ResolveChoice
27 ## @args: [choice = "make_cols", transformation_ctx = "resolvechoice2"]
28 ## @return: resolvechoice2
29 ## @inputs: [frame = applymapping1]
30 resolvechoice2 = ResolveChoice.apply(frame = applymapping1, choice = "make_cols", transformation_ctx = "resolvechoice2")
31 ## @type: DropNullFields
32 ## @args: [transformation_ctx = "dropnullfields3"]
33 ## @return: dropnullfields3
34 ## @inputs: [frame = resolvechoice2]
35 dropnullfields3 = DropNullFields.apply(frame = resolvechoice2, transformation_ctx = "dropnullfields3")
36 ## @type: DataSink
37 ## @args: [catalog_connection = "IRIS1", connection_options = {"dbtable": "elb_logs", "database": "USER"},
38 ## @return: datasink4
39 ## @inputs: [frame = dropnullfields3]
40 datasink4 = glueContext.write_dynamic_frame.from_jdbc_conf(frame = dropnullfields3, catalog_connection = "IRIS1", connection_options = {"dbtable": "elb_logs", "database": "USER"}, transformation_ctx = "datasink4")
41 job.commit()

```

Logs Schema

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Almost there!!! We just need to make one simple change to the script in order to load data into IRIS.

Adjusting the script

The script that Glue generated for us uses [AWS Glue Dynamic Frame](#) - AWS proprietary extension to Spark. While it provides some benefits for ETL jobs it also ensures that you can't write data to any database that AWS don't have managed service offering for.

Good news - at the point of writing the data to the database all the benefits of Dynamic Dataframe such as no schema enforcement for "dirty" data are not required anymore (at the point of writing data is presumed to be "clean") and we can easily convert Dynamic Dataframe to Spark native Dataframe that is not limited to AWS managed targets and can work with IRIS.

So the line we need to change is line #40 on the picture above. One before last.

Here is the change we need to make:

```

#datasink4 = glueContext.write_dynamic_frame.from_jdbc_conf(frame = dropnullfields3,
catalog_connection = "IRIS1", connection_options = {"dbtable": "elb_logs", "database"
: "USER"}, transformation_ctx = "datasink4")
dropnullfields3.toDF().write \
    .format("jdbc") \
    .option("url", "jdbc:IRIS://172.30.0.196:51773/USER/") \
    .option("dbtable", "orders") \
    .option("user", irisUsername) \
    .option("password", irisPassword) \
    .option("isolationlevel", "NONE") \
    .save()

```


Where irisUsername and irisPassword are the username and passwords for your IRIS JDBC connection.

Note: storing passwords in the source code is a big No No! We'll encourage you to use tools like [AWS Secrets Manager](#) for that, but going into this level of the security details is beyond the scope of this article. Here is the good [article](#) on using AWS Secrets Manager with AWS Glue.

Now hit "Run Job" button, sit back and relax while AWS Glue is doing ETL for you.

Well... more than likely you'll hit some errors at first... We all know how it works. A typo here, a wrong port in security group there... AWS Glue uses CloudWatch to store all the execution and error logs. Browse /aws-glue/jobs/error and /aws-glue/jobs/output log groups to identify what went wrong.

Happy ETing in the cloud!

-Anton

[#AWS](#) [#Best Practices](#) [#Big Data](#) [#Cloud](#) [#Databases](#) [#Python](#) [#SQL](#) [#InterSystems IRIS](#)

Source URL: <https://community.intersystems.com/post/using-aws-glue-intersystems-iris>