I must preface this post with the posit that if you’re doing anything interactive() with Amazon Athena you should seriously consider just using their [free ODBC drivers](https://docs.aws.amazon.com/athena/latest/ug/connect-with-odbc.html) as it’s the easiest way to wire them up to R DBI- and tidyverse-wise. Drop a note in the comments if you don’t know the incantations for repackaging the provided Linux ODBC drivers to work on your flavor of Linux.

*However*…

There are times—say, when you’re trying to stand up an R service in your kubernetes cluster which bridges data in Athena to analyses & visualizations in R—when ODBC drivers can be more of a hindrance than help and JDBC is the path of least resistance.

Sure, there’s the in-CRAN AWR.Athena package but it’s a fairly constrained and low-feature RJDBC shim which gets the basic job done but not much more.

Enter:

* metis.jars
* metis
* metis.tidy

A trio of packages which aims to make it super-straightforward to wire up R to Amazon Athena when ODBC is not available.

**Why Three Packages?**

For starters, there are CRAN hopes for the metis-trio and one key component of that is separating out the JARs into one package (metis.jars) and actual functionality into others (metis and metis.tidy). We’ll see how the CRAN attempt goes since the JAR package weighs in at sufficient weight to warrant a NOTE. The packaging of the driver reduces the need for you to pre-load the JAR (locally or into, say, a Docker image) or perform a package-initiated download-dance like AWR.Athena does (which I still don’t understand why that hasn’t kicked it out of CRAN the way it does it but ¯\\_(ツ)\_/¯).

metis.jars also has three helper functions which do some (basic) fun things:

library(metis.jars)

simba\_driver\_version()

## [1] "02.00.06.1008"

athena\_supported\_types()

## [1] "BOOLEAN" "TINYINT" "SMALLINT" "INT" "INTEGER"

## [6] "BIGINT" "REAL" "FLOAT" "DOUBLE" "DECIMAL"

## [11] "DATE" "TIMESTAMP" "BINARY" "VARBINARY" "CHAR"

## [16] "VARCHAR" "STRING" "ARRAY" "MAP" "ROW"

## [21] "STRUCT"

metis\_jar\_path()

## [1] "/Library/Frameworks/R.framework/Versions/3.5/Resources/library/metis.jars/java/AthenaJDBC42\_2.0.6.jar"

The first uses the rJava interface to directly query the version (since Amazon seems to update the Simba JAR twice a year). By separating out the JAR into a separate package, updates can be made to the other two sibling packages more frequently without crushing CRAN’s disk space. metis.jars is also versioned to the included JAR so configuration management will be easier for folks.

The reason for the second type-lister function is that there’s hope Amazon will add support for *all* Presto data types, especially IPADDRESS. It, again, performs JDBC driver introspection to collect the supported types.

Finally, the third function abstracts the JAR location from the metis package or even *your own* interface package should you choose to depend on it.

**OK, But Why Not Just Two?**

The metis package is a more functional RJDBC superclass of a DBI wrapper than AWR.Athena. One thing it does that its CRAN cousin cannot is handle BIGINTs properly:

library(metis)

dbConnect(

metis::Athena(),

Schema = "sampledb",

AwsCredentialsProviderClass = "com.simba.athena.amazonaws.auth.PropertiesFileCredentialsProvider",

AwsCredentialsProviderArguments = path.expand("~/.aws/athenaCredentials.props")

) -> con

dbGetQuery(con, "

SELECT

CAST('chr' AS CHAR(4)) achar,

CAST('varchr' AS VARCHAR) avarchr,

CAST(SUBSTR(timestamp, 1, 10) AS DATE) AS tsday,

CAST(100.1 AS DOUBLE) AS justadbl,

CAST(127 AS TINYINT) AS asmallint,

CAST(100 AS INTEGER) AS justanint,

CAST(100000000000000000 AS BIGINT) AS abigint,

CAST(('GET' = 'GET') AS BOOLEAN) AS is\_get,

ARRAY[1, 2, 3] AS arr1,

ARRAY['1', '2, 3', '4'] AS arr2,

MAP(ARRAY['foo', 'bar'], ARRAY[1, 2]) AS mp,

CAST(ROW(1, 2.0) AS ROW(x BIGINT, y DOUBLE)) AS rw,

CAST('{\"a\":1}' AS JSON) js

FROM elb\_logs

LIMIT 1

") %>%

dplyr::glimpse()

## Observations: 1

## Variables: 13

## $ achar "chr "

## $ avarchr "varchr"

## $ tsday 2014-09-29

## $ justadbl 100.1

## $ asmallint 127

## $ justanint 100

## $ abigint 100000000000000000

## $ is\_get TRUE

## $ arr1 "1, 2, 3"

## $ arr2 "1, 2, 3, 4"

## $ mp "{bar=2, foo=1}"

## $ rw "{x=1, y=2.0}"

## $ js "\"{\\\"a\\\":1}\""

~~Presto~~Athena arrays and maps and rows and JSON come across as characters from the Athena driver and they’re formatted so badly that there’s little hope of full R support for list columns for them. *But*, you do get real, big integers with metis along with full support for all other current Athena types.

R folk who may be users of the old, standalone metis package need to be aware of some things.

First, dbConnect() has breaking changes. The snake\_case names that still exist in the higher-level athena\_jdbc() function are gone. In exchange for this pain, you now have full naming-parity with all the Athena JDBC connection properties and can more easily use alternate credential providers.

The metis package also makes it easier to see documentation for *all* available Athena connection properties since it has a vignette with a descriptive table of all of them .

There is also nascent support for the “streaming API” (TLDR: faster result set downloads) but that won’t be fully tested until some AWS policy tweaks happen this week.

**Gotcha. But, Why Not Just Two?**

As awesome as it is (including base Docker image support) the tidyverse is not without overhead in terms of compilation time and dependencies, both of which are especially painful on Linux systems and some Docker environments. You can absolutely get by with some well-crafted SQL and JDBC and the thinner the image the easier it is to deploy and scale.

But! The tidyverse is *so helpful* that ensuring smooth support for Athena is critical. On its own, metis wires up to dplyr/dbplyr fine, but by providing (in metis.tidy) some enhanced db\_data\_type() support (primarily for BIGINT) and some extra in sql\_translate\_env() )for those of us who continue to mindlessly use R-only verbs like grep() or as.POSIXct() in non-R contexts) we can level-up interactive() use and tidyverse-infused service use:

library(metis.tidy)

library(dbplyr)

library(dplyr)

metis::dbConnect(

metis::Athena(),

Schema = "sampledb",

AwsCredentialsProviderClass = "com.simba.athena.amazonaws.auth.PropertiesFileCredentialsProvider",

AwsCredentialsProviderArguments = path.expand("~/.aws/athenaCredentials.props")

) -> con

elb\_logs <- tbl(con, "elb\_logs")

filter(elb\_logs, grepl("20", elbresponsecode)) %>%

mutate(

tsday = as.Date(substring(timestamp, 1L, 10L)),

host = url\_extract\_host(url),

proto\_version = regexp\_extract(protocol, "([[:digit:]\\.]+)"),

) %>%

select(tsday, host, receivedbytes, requestprocessingtime, proto\_version) %>%

head(1) %>%

glimpse()

## Observations: ??

## Variables: 5

## Database: AthenaConnection

## $ tsday 2014-09-29

## $ host "www.abcxyz.com"

## $ receivedbytes 0

## $ requestprocessingtime 9.5e-05

## $ proto\_version "1.1"

**FIN**

A fairly big impetus for this radical refactoring was the need to use the Athena JDBC interface in R at $DAYJOB in a serverless context. So, if I/we needed it, others may as well. All three packages have tests (that work with my personal Athena setup which is easily replicated since it’s just the default schema & table you get when you enable Athena), pass CRAN checks and will be live in a real production environment by the time you read this.

Note that I do have CRAN plans for these three amigos, but all three packages will need to go in at the same time and I need to get tests into- and prove tests are live in Travis before submitting.