It was probably not difficult to discern that I’m fairly excited about the Apache Drill 1.15.0 release. I’ve rounded out most of the existing corners for it in preparation for a long-overdue CRAN update and have been concentrating on two helper features: configuring & launching Drill embedded Docker containers and auto-generation of Drill CTAS queries.

**Drill Docker Goodness**

Starting with version 1.14.0, Apache provides Drill Docker images for use in experimenting/testing/building-off-of. They run Drill in single node standalone mode so you’re not going to be running this in “production” (unless you have light or just personal workloads). Docker is a great way to get to know Drill if you haven’t already played with it since you don’t have do do much except run the Docker image.

I’ve simplified this even more most excellent stevedore package which adds a robust R wrapper to the Docker client without relying on any heavy external dependencies such as reticulate. The new drill\_up() function will auto-fetch the latest Drill image and launch a container so you can have a running Drill instance with virtually no effort on your part.

Just running the vanilla image isn’t enough since your goal is likely to do more than work with the built-in cp data source. The default container launch scenario also doesn’t hook up any local filesystem paths to the container so you really can’t do much other than cp-oriented queries. Rather than make you do all the work of figuring out how to machinate Docker command line arguments and manually configure a workspace that points to a local filesystem area in the Drill web admin GUI the drill\_up() function provides a data\_dir argument (that defaults to the getwd() where you are in your R session) which will then auto-wire up that path into the container **and** create a dfs.d workspace which auto-points to it for you. Here’s a sample execution:

library(sergeant)

library(tidyverse)

library(stevedore)

dr <- drill\_up(data\_dir = "~/Data")

## Drill container started. Waiting for the service to become active (this may take up to 30s).

## Drill container ID: f02a11b50e1647e44c4e233799180da3e907c8aa27900f192b5fd72acfa67ec0

You can use dc$stop() to stop the container or use the printed container id to do it from the command line.

**Taking the Tedium out of CTAS**

Prior to 1.14.0 sergeant had no way to accurately, precisely tell data types of the columns coming back since the REST API didn’t provide them (as noted in the previous Drill post). Now, it *did* rely on the JSON types to create the initial data frames *but* id also did something \*\*kinda horribad\*: it ran readr::type\_convert() on the result sets . Said operation had the singular benefit of auto-converting CSV/CSVH/TSV/PSV/etc data to something sane without having to worry about writing lengthy CTAS queries (at the expense of potentially confusing everyone, though that didn’t seem to happen).

With 1.15.0, the readr::type\_convert() crutch is gone, which results in less-than-helpful things like this when you have delimiter-separated values data:

# using the Drill container we just started above

write\_csv(nycflights13::flights, "~/Data/flights.csvh")

con <- src\_drill("localhost")

tbl(con, "dfs.d.`flights.csvh`") %>%

glimpse()

## Observations: ??

## Variables: 19

## Database: DrillConnection

## $ year "2013", "2013", "2013", "2013", "2013", "2013", "2013", "2013…

## $ month "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "…

## $ day "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "…

## $ dep\_time "517", "533", "542", "544", "554", "554", "555", "557", "557"…

## $ sched\_dep\_time "515", "529", "540", "545", "600", "558", "600", "600", "600"…

## $ dep\_delay "2", "4", "2", "-1", "-6", "-4", "-5", "-3", "-3", "-2", "-2"…

## $ arr\_time "830", "850", "923", "1004", "812", "740", "913", "709", "838…

## $ sched\_arr\_time "819", "830", "850", "1022", "837", "728", "854", "723", "846…

## $ arr\_delay "11", "20", "33", "-18", "-25", "12", "19", "-14", "-8", "8",…

## $ carrier "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", "AA", "…

## $ flight "1545", "1714", "1141", "725", "461", "1696", "507", "5708", …

## $ tailnum "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N39463", "…

## $ origin "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA", "JFK"…

## $ dest "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD", "MCO"…

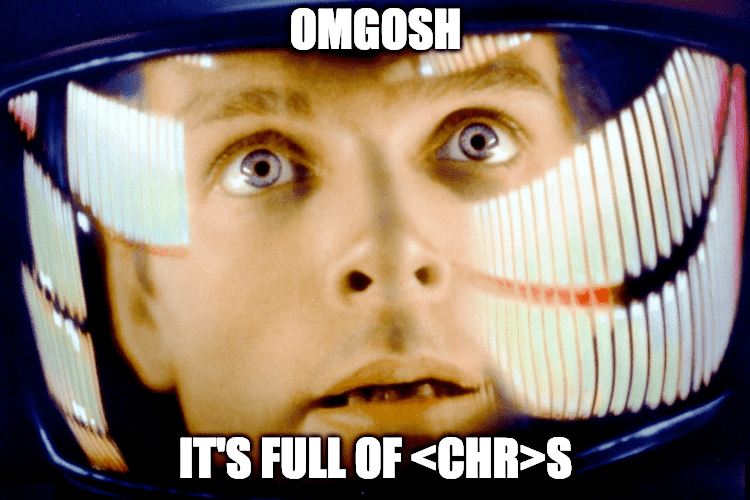
## $ air\_time "227", "227", "160", "183", "116", "150", "158", "53", "140",…

## $ distance "1400", "1416", "1089", "1576", "762", "719", "1065", "229", …

## $ hour "5", "5", "5", "5", "6", "5", "6", "6", "6", "6", "6", "6", "…

## $ minute "15", "29", "40", "45", "0", "58", "0", "0", "0", "0", "0", "…

## $ time\_hour "2013-01-01T10:00:00Z", "2013-01-01T10:00:00Z", "2013-01-01T1…

[](https://rud.is/b/2019/01/09/on-the-road-to-0-8-0-some-additional-new-features-coming-in-the-sergeant-package/chrs/)

So the package does what it finally should have been doing all along. But, as noted, that’s not great if you just wanted to quickly work with a directory of CSV files. In theory, you’re supposed to use Drill’s CREATE TABLE AS then do a bunch of CASTS and TO\_s to get proper data types. But who has time for that?

David had a *stellar* idea, *might sergeant be able to automagically create CTAS statements from a query?*. Yes. Yes it just might be able to do that with the new ctas\_profile() function.

Let’s pipe the previous tbl() into ctas\_profile() and see what we get:

tbl(con, "dfs.d.`flights.csvh`") %>%

ctas\_profile() %>%

cat()

-- \*\* Created by ctas\_profile() in the R sergeant package, version 0.8.0 \*\*

CREATE TABLE CHANGE\_\_\_\_ME AS

SELECT

CAST(`year` AS DOUBLE) AS `year`,

CAST(`month` AS DOUBLE) AS `month`,

CAST(`day` AS DOUBLE) AS `day`,

CAST(`dep\_time` AS DOUBLE) AS `dep\_time`,

CAST(`sched\_dep\_time` AS DOUBLE) AS `sched\_dep\_time`,

CAST(`dep\_delay` AS DOUBLE) AS `dep\_delay`,

CAST(`arr\_time` AS DOUBLE) AS `arr\_time`,

CAST(`sched\_arr\_time` AS DOUBLE) AS `sched\_arr\_time`,

CAST(`arr\_delay` AS DOUBLE) AS `arr\_delay`,

CAST(`carrier` AS VARCHAR) AS `carrier`,

CAST(`flight` AS DOUBLE) AS `flight`,

CAST(`tailnum` AS VARCHAR) AS `tailnum`,

CAST(`origin` AS VARCHAR) AS `origin`,

CAST(`dest` AS VARCHAR) AS `dest`,

CAST(`air\_time` AS DOUBLE) AS `air\_time`,

CAST(`distance` AS DOUBLE) AS `distance`,

CAST(`hour` AS DOUBLE) AS `hour`,

CAST(`minute` AS DOUBLE) AS `minute`,

TO\_TIMESTAMP(`time\_hour`, 'FORMATSTRING') AS `time\_hour` -- \*NOTE\* You need to specify the format string. Sample character data is: [2013-01-01T10:00:00Z].

FROM (SELECT \* FROM dfs.d.`flights.csvh`)

-- TIMESTAMP and/or DATE columns were detected.

Drill's date/time format string reference can be found at:

--

--

There’s a parameter for the new table name which will cause the CHANGE\_\_\_\_ME to go away and when the function finds TIMESTAMP or DATE fields it knows to switch to their TO\_ cousins and gives sample data with a reminder that you need to make a format string (I’ll eventually auto-generate them unless someone PRs it first). And, since nodoby but Java programmers remember Joda format strings (they’re different than what you’re used to) it provides a handy link to them if it detects the presence of those column types.

Now, we don’t need to actually create a new table (though converting a bunch of CSVs to Parquet is likely a good idea for performance reasons) to use that output. We can pass most of that new query right to tbl():

tbl(con, sql("

SELECT

CAST(`year` AS DOUBLE) AS `year`,

CAST(`month` AS DOUBLE) AS `month`,

CAST(`day` AS DOUBLE) AS `day`,

CAST(`dep\_time` AS DOUBLE) AS `dep\_time`,

CAST(`sched\_dep\_time` AS DOUBLE) AS `sched\_dep\_time`,

CAST(`dep\_delay` AS DOUBLE) AS `dep\_delay`,

CAST(`arr\_time` AS DOUBLE) AS `arr\_time`,

CAST(`sched\_arr\_time` AS DOUBLE) AS `sched\_arr\_time`,

CAST(`arr\_delay` AS DOUBLE) AS `arr\_delay`,

CAST(`carrier` AS VARCHAR) AS `carrier`,

CAST(`flight` AS DOUBLE) AS `flight`,

CAST(`tailnum` AS VARCHAR) AS `tailnum`,

CAST(`origin` AS VARCHAR) AS `origin`,

CAST(`dest` AS VARCHAR) AS `dest`,

CAST(`air\_time` AS DOUBLE) AS `air\_time`,

CAST(`distance` AS DOUBLE) AS `distance`,

CAST(`hour` AS DOUBLE) AS `hour`,

CAST(`minute` AS DOUBLE) AS `minute`,

TO\_TIMESTAMP(`time\_hour`, 'yyyy-MM-dd''T''HH:mm:ssZ') AS `time\_hour` -- [2013-01-01T10:00:00Z].

FROM (SELECT \* FROM dfs.d.`flights.csvh`)

")) %>%

glimpse()

## Observations: ??

## Variables: 19

## Database: DrillConnection

## $ year 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2…

## $ month 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…

## $ day 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…

## $ dep\_time 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558, 558, 5…

## $ sched\_dep\_time 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600, 600, 6…

## $ dep\_delay 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -1, 0, -…

## $ arr\_time 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849, 853, …

## $ sched\_arr\_time 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851, 856, …

## $ arr\_delay 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -14, 31,…

## $ carrier "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", "AA", "…

## $ flight 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, 49, 71,…

## $ tailnum "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N39463", "…

## $ origin "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA", "JFK"…

## $ dest "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD", "MCO"…

## $ air\_time 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, 158, 34…

## $ distance 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733, 1028, …

## $ hour 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6…

## $ minute 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, 0, 0, 0…

## $ time\_hour 2013-01-01 10:00:00, 2013-01-01 10:00:00, 2013-01-01 10:00:0…

Ahhhh… Useful data types. (And, see what I mean about that daft format string? Also, WP is mangling the format string so add a comment if you need the actual string.)