

Introducing Athenadriver: An Open Source Amazon Athena Database Driver for Go



Henry Fuheng Wu, Raymond Won, Nick Cobb, Mingjie Lai, and Matt Ranney

Data analytics play a critical part in Uber's decision making, driving and shaping all aspects of the company, from improving our products to generating insights that inform our business. To ensure timely and accurate analytics, the aggregated, anonymous data that powers these analytics needs to be easily searchable.

Due to the scale and complexity of our business, data is often heterogeneous and stored in different data stores, including [MySQL](#), [Presto](#), [Schemaless](#), and [AresDB](#). To address these disparities, we unified these data stores using a datasource connector written in [Go](#) and powered by the language's built-in [database package](#), providing a uniform way to query necessary data.

At [Uber ATG](#), however, we are following Uber's existing strategy and moving to a hybrid cloud solution that leverages some services from Amazon AWS, which offers gains in cost efficiency,

performance, and maintainability. As we scaled this solution, it became more difficult to query data when we needed to generate reports or run machine learning tasks. While Uber's in-house business intelligence tools can allow for easy sharing of data between systems, for instance, our [machine learning and data science platforms](#), they did not support AWS cloud data queries.

We decided to use Amazon Athena, an interactive query service, to fill this gap, enabling our existing business intelligence tools to query AWS cloud data. To facilitate communication between Uber's business intelligence tools and AWS Athena, we built [Athenadriver](#), a database driver for Amazon Athena that supports Go's built-in database/SQL framework.

Recently, we open sourced [Athenadriver](#), allowing others in the community to benefit from our solution for their own large-scale data analytics needs. We hope you find Athenadriver useful and will consider contributing features and suggestions of your own so we can continue to enhance the project for users worldwide.

How Athenadriver works

Amazon Athena makes it easy to analyze data in S3, AWS object storage service, using standard SQL. [Presto](#), the distributed query engine underlying Athena, handles SQL parsing, workload distribution, and execution. And unlike other traditional relational database management systems (RDBMS) like MySQL and PostgreSQL, Athena was designed for querying large data stores quickly.

In order to support communication between Uber's business intelligence tools, many of which are written in Go, and Athena, a service that does not currently support integrations with Go, we built Athenadriver, an open source driver that serves as a bridge. Leveraging Athenadriver, users can access Athena through Uber's existing business intelligence frontend, utilizing the same interface as our existing drivers for MySQL, AresDB, and other databases.

Importantly, Athenadriver allows our teams to maintain a uniform environment between databases. Users simply select S3 cloud data as the query target datastore type, and results are visualized on their business intelligence dashboard of choice.

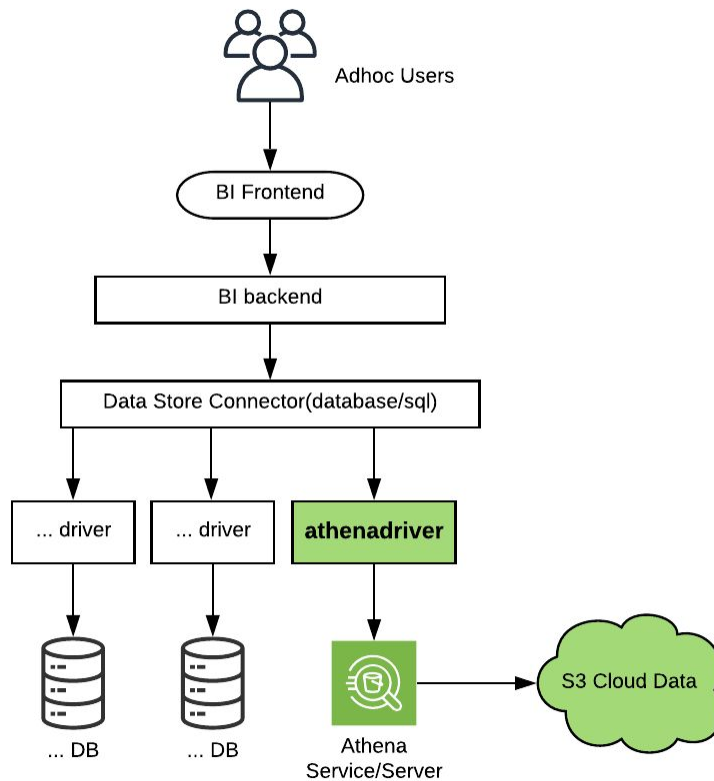


Figure 1: Athenadriver sends queries from our Go-based business intelligence tools to Athena, Amazon's AWS interactive query service.

Underneath this familiar-looking shell, Athenadriver calls Athena's Go SDK to interact with the actual Athena service, sending SQL requests and fetching query result sets. An Athenadriver query consists of three distinct steps:

1. The client sends an SQL query from the Go datasource connector to Athenadriver, which relays this request to Athena. At this stage, Athena merely receives the query request and adds the task to its execution queue. Athena assigns the task a unique query ID string which is immediately returned to Athenadriver.
2. Athenadriver uses this query ID string to periodically request the status of the SQL query execution task while it sits in Athena's queue. If the task is still in the queue, this step repeats.
3. When the status indicates the SQL query has been executed successfully, Athenadriver sends a second SQL query to pull the initial query's results. Because we are working with big data, the query results are sometimes so huge that returning them all at once might block the network or use up too much memory. In these cases, Athenadriver sends multiple result-fetching queries to fetch data page by page.

Figure 2, below, shows all three steps in an Athenadriver query's life cycle. Athena uses Presto to query S3 source data and then stores the results in another S3 result set bucket.

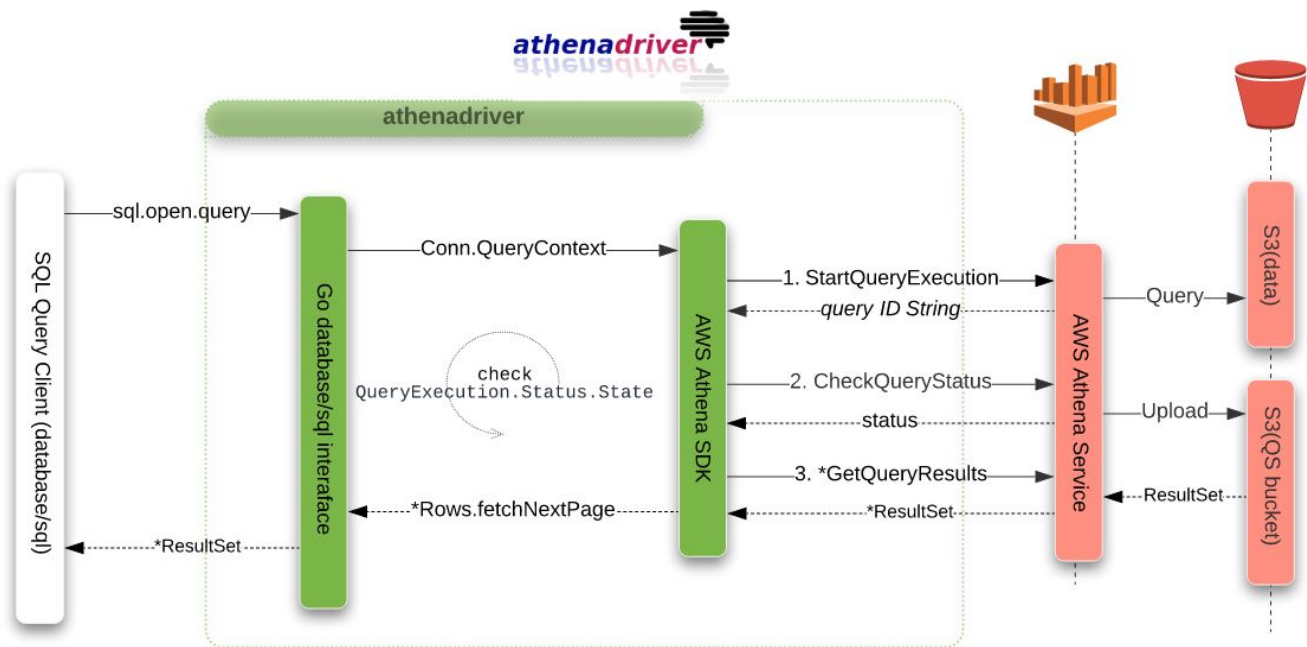


Figure 2: During a query life cycle, Athenadriver relays a query to Athena, then periodically checks Athena for the status of the query. When Athena processes the query, Athenadriver sends a second query to gather results.

As Figure 2 shows, querying Athena actually makes for a very complicated workflow. In a normal Go database driver like MySQL or PostgreSQL, a query is completed in just one step and with one sub-query, but in Athena's case, this process takes three steps and at least three sub-queries to Athena services. Athenadriver simplifies this querying process, improving productivity for engineering teams.

Athenadriver enhances and extends the basic Athena Go SDK, inherits the basic feature set of Go's built-in database/SQL framework, and addresses some of the Athena Go SDK's limitations. Moreover, it also includes advanced features such as support for multiple AWS authentication methods, Athena workgroup and tagging creation, and read-only mode.

Querying with workgroups and tags

Athenadriver fully supports Athena's workgroups and tags, which can be used for functions such as cost tracking, user management, and query history management. While these features fall outside the scope of the standard Go database/SQL framework, Athenadriver allows users to leverage both features within existing business intelligence tools. Athenadriver uses a new format of [data source name](#) (DSN) to identify unique databases. A standard DSN (used by most databases) has the following format:

```
<driver>://<username>:<password>@<host>:<port>/<database>
```

The new format we invented for Athena DSN looks like this:

```
s3://<user>@<result_set_bucket>?<database_properties_key_value_pairs>
```

The `database_properties_key_value_pairs` includes credentials, workgroups, tags, session, and driver configuration. Athenadriver then encapsulates all these elements into a single configuration object, a means of storing parameters for reuse, and uses it to call the DSN string.

We then set the properties of the configuration object, serialize it to the Athena DSN string, and pass it to the Go database/SQL API. Athenadriver deserializes the DSN string back into a configuration object, allowing it to receive the pertinent information (including workgroups and tags). With this information, Athenadriver can invoke the Athena Go SDK API to get existing workgroups and tags from Athena on the server side. It can also create workgroups and tags remotely on Athena's server side.

At Uber, we use Athenadriver to automate workgroup and tag creation, as well as track costs. Since Amazon runs Athena as a pay-per-query service, workgroups and tags help us fine-tune our queries ahead of time, preventing unnecessarily large queries. Athenadriver also helps us track our expenditures. When somebody runs an overly-broad query, our cost efficiency system can pinpoint that user and notify them.

Ensuring data consistency with read-only mode

Athenadriver also supports read-only mode, allowing read-only mode allows certain users to retrieve but not override information from their Athena database. When the system administrator turns on this feature, any writing and modification to the database will raise an error, preventing accidental data changes.

Normally used for analytic queries, Athena also allows users to modify and create databases. This kind of writing to database operation does not create new files in the original S3 datasource. Instead, it creates intermediate files in the output bucket.

Although it is possible to disable write access in the S3 bucket policies, Athenadriver's read-only mode allows database administrators to directly control the database write permission based on client information since it operates on the client-side of the system. This capability allows database administrators to create local user groups with different access levels.

As an example of the value of Athenadriver's read-only mode, the `queryAthena()` function, below, accepts `someUser` as the input.

```

1 func queryAthena(someUser) {
2     // 1. Set AWS Credential in Driver Config.
3     conf, _ := drv.NewDefaultConfig("s3://query-results-bucket/",
4         "us-east-2", "AccessID", "SecretAccessKey")
5     if someUser == "data scientist"{
6         conf.SetReadOnly(true)
7     } else if someUser == "database administrator"{
8         conf.SetReadOnly(false)
9     }
10    // 2. Open Connection.
11    dsn := conf.Stringify()
12    db, _ := sql.Open(drv.DriverName, dsn)
13    // 3. Create Table with CTAS statement
14    rows, err := db.QueryContext(context.Background(),
15        "CREATE TABLE sampled.elb_logs_new AS SELECT * FROM sampled.elb_logs")
16    defer rows.Close()
17 }

```

If someUser is a data scientist (as seen on line 5), then they can only retrieve information because read-only mode is enabled. If someUser is a database administrator (as shown on line 7), then read-only mode is disabled and they can freely modify the database.

Driving forward






The above covers only three of the many features that make Athenadriver useful, but as stated earlier, Athenadriver also extends the Go database/SQL framework and addresses many of the Amazon Athena Go SDK's limitations. Athenadriver can greatly simplify Athena query code by encapsulating query and access complexities and seamlessly handling various edge cases so engineers can safely replace hundreds of lines of Athena Go SDK code with less than ten lines of code for Athenadriver.

Athenadriver even supports [a few undocumented Athena features](#), such as additional data types. We open sourced Athenadriver along with our companion utility tool [AthenaReader](#) in February of 2020 to help developers dealing with similar issues in their own organizations. As we began integrating Athena and Uber's business intelligence tools, we realized an Athena driver compatible with Go's standard library was not only important but required—and we could not find one that met our needs. As such, we imagine many teams combining internal Go-based services with AWS will find Athenadriver useful.

Since open sourcing Athenadriver, the project has been cloned tens of thousands of times and already helped numerous developers and users realize the full power of Athena. We look forward to continuing to maintain the project, addressing user issues as they occur and updating Athenadriver as Go and Athena evolve.

We are also considering open sourcing more features and utilities similar to [Athenareader](#), as well as drivers in other languages, and as such, we welcome feedback and contributions to [Athenadriver's github repository](#). We look forward to collaborating with you!

This article's header image includes the [Go gopher logo](#), which was created by Rene French and is licensed under [Creative Commons Attribution 3.0](#).

	<p>Henry Fuheng Wu</p> <p>Henry Fuheng Wu is a software engineer on the Uber ATG Infrastructure team. He likes playing basketball and guitar when there is no code to write. As a software developing veteran and hard core technologist, he has found peace of mind on Uber ATG's Infrastructure team.</p>
	<p>Raymond Won</p> <p>Raymond Won is the engineering manager for the Uber ATG Data Infrastructure team. His team is responsible for the ATG Analytics Platform, workflow management system, storage systems, and infrastructure efficiency. He enjoys playing basketball and taking long walks with his kids and their dog in his spare time.</p>
	<p>Nick Cobb</p> <p>Nick Cobb is a Senior Manager on the Uber ATG Infrastructure team, which is responsible for Uber ATG's SRE, cloud, data center, compute, and data infrastructure domains.</p>
	<p>Mingjie Lai</p> <p>Mingjie Lai is a staff software engineer on the Uber ATG Infrastructure team. He created the Uber ATG analytics platform and is leading the technology and innovation of ATG Infrastructure. He likes to play soccer and watch TV with his family in his spare time.</p>
	<p>Matt Ranney</p> <p>Matt Ranney is a senior staff software engineer on the Uber ATG Simulation team. He is also a member of the Uber Open Source Software Committee. You can find him at many conferences like GOTO, YOW, and Surge.</p>