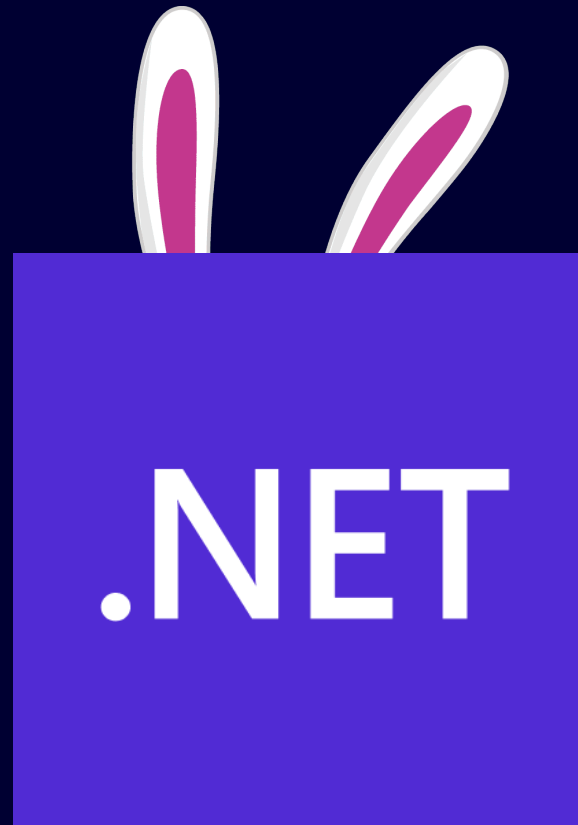


Trino C# .NET Client

Trino Conference Winter 2024



What is it?



- Trino .NET (C#) Client Library
 - .NET Standard 2.0
 - Compatible with .NET Framework 4.7.2 & .NET Core 2.0
- Open source from Microsoft
- ADO.NET implementation
- nuget packaging

.NET implementation	Version support
.NET and .NET Core	2.0, 2.1, 2.2, 3.0, 3.1, 5.0, 6.0, 7.0, 8.0, 9.0
.NET Framework ¹	4.6.1 ² , 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Origins

- 6 years of iterating...
- Windows/ASP.NET

George Fisher
Oleg Savin
Ruslan Kartokhin
Ishan Patwa
Dimitry Berger

Special thanks to
David Lao
Xiaotong Zhang
Pruthvi Prodduturi

Design Objectives

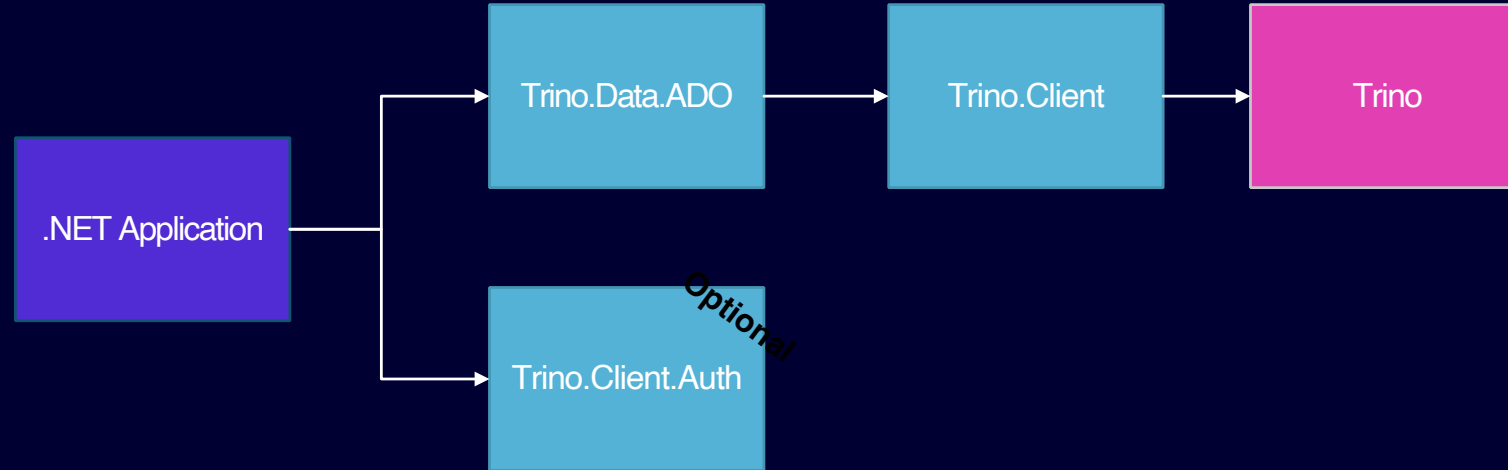


- **Responsive/async** (fast return of rows for both short and long running queries).
- **.NET Type Support** (with complex types, date time precision, and System.Data types.)
- **Authentication** that is customizable and flexible (to support arbitrary custom token refresh, or unique cloud requirements).
- **Full ADO.NET** implementation.
- **Streaming** with read-ahead into customizable buffer size to allow for no-wait client paging.
- **Session support.**
- **Parameterized query support.**
- **Reduce DLL hell** no dependencies outside of .NET core except Newtonsoft.Json and Microsoft.Extensions.Logging. Authentication in separate library.
- **Alignment with Trino Java client**, similar class structure.

Usage/Project Structure

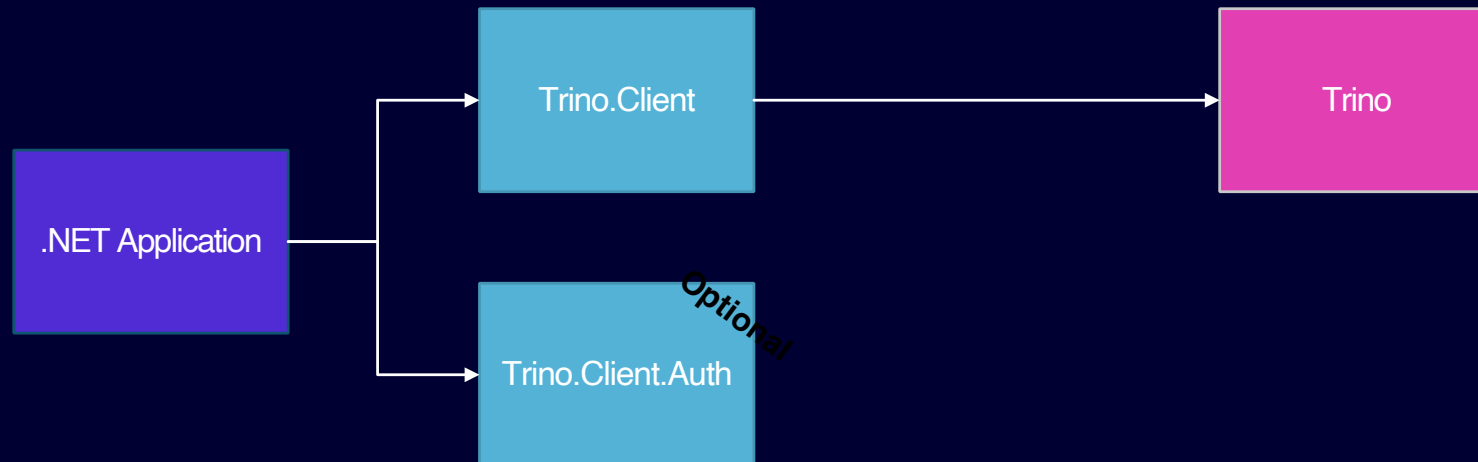


ADO.NET Approach



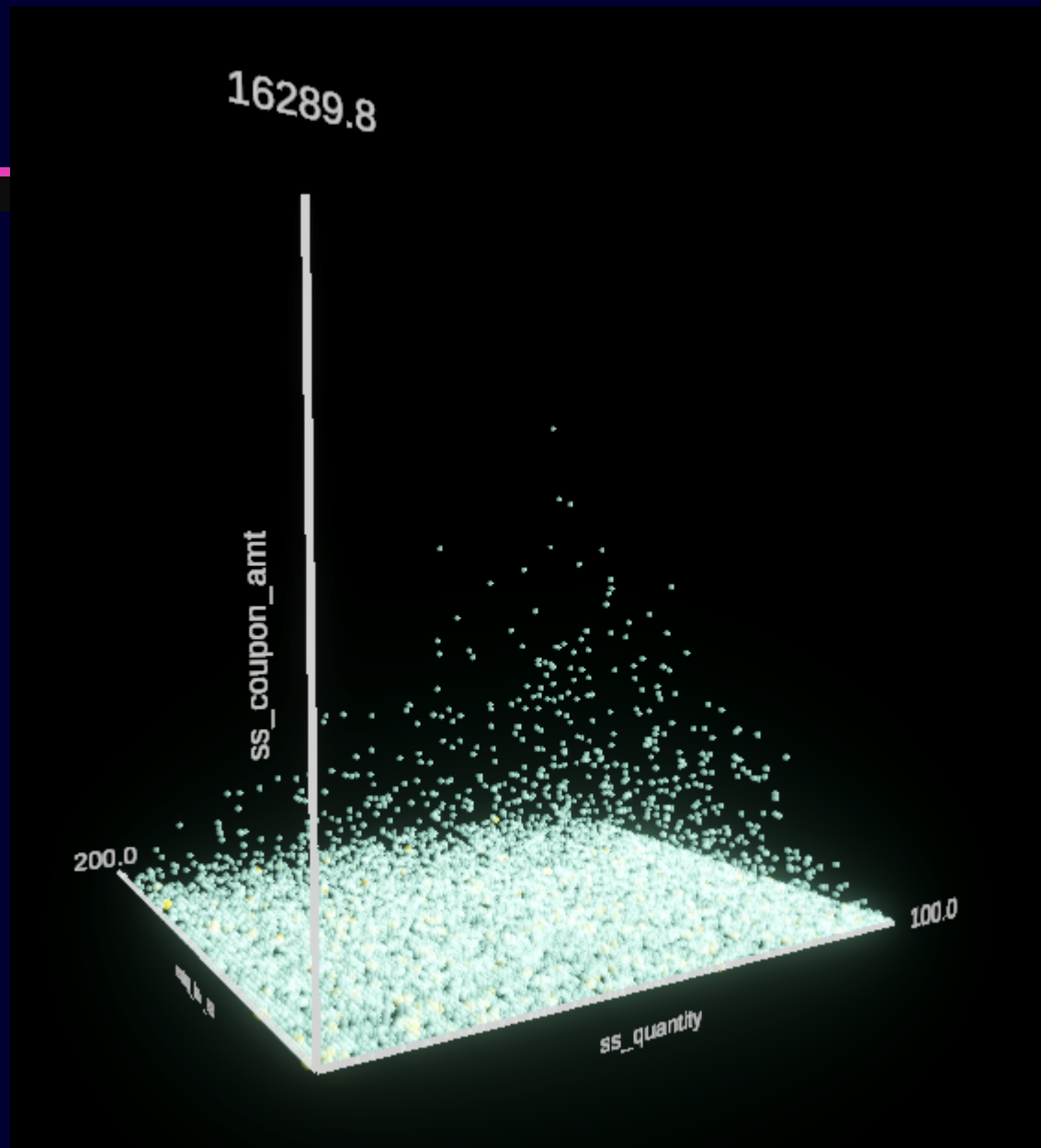
Direct Approach

Low level



Demo

- Trino .NET demo



.NET

ADO.NET Library: Trino.Data.ADO



.NET

- What is ADO.NET?
 - DbConnection, DbCommand, DbDataReader
- Why? Integration, familiarity, simplicity
- Connection string
 - Same properties as `TrinoConnectionProperties`
 - Authentication
- Test Connection
 - Off by default
 - Can be turned on for `Open()`
- Schema support
 - `GetSchema()`
 - `DataTable GetSchema(string collectionName, string[] restrictionValues)`
 - Catalogs, databases, schemas/schemata, tables, columns, views, functions, sessions
- Custom types: `BigDecimal` and `YearToMonth`

```
using (TrinoConnection tc = new TrinoConnection())
{
    tc.ConnectionString =
        $"catalog=delta;schema=nyc;host={demoClusterHost};auth=TrinoJWTAuth;AccessToken={token}";
}
```

```
using (var connection = new TrinoConnection(properties))
{
    connection.Open();
    using (var command = new TrinoCommand(connection,
        query))
    using (var reader = command.ExecuteReader())
    {
        float x = (float)reader.GetDouble(0);
        float y = (float)reader.GetDecimal(1);
        float z = (float)reader.GetDecimal(2);
        ...
    }
}
```

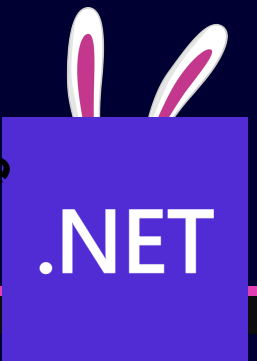
Trino.Client

- How to use

Low level

```
ClientSession session = new ClientSession(  
    sessionProperties: new ClientSessionProperties()  
    {  
        Server = new Uri("http://localhost:8080/")  
    });
```

```
DataTable dt = await(  
    await RecordExecutor.Execute(  
        session: session,  
        statement: "select * from tpch.tiny.customer"  
    )  
).BuildDataTableAsync();  
Console.WriteLine($"Read {dt.Rows.Count} rows");
```



.NET

Trino.Client

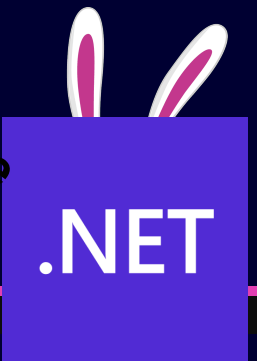
- How to use

Low level

```
ClientSession session = new ClientSession(  
    sessionProperties: new ClientSessionProperties()  
    {  
        Server = new Uri("http://localhost:8080/")  
    });
```

```
RecordExecutor records = await RecordExecutor.Execute(  
    session: session,  
    statement: "select * from tpch.tiny.customer"  
).ConfigureAwait(false);
```

```
foreach (var row in records)  
{  
    Console.WriteLine(string.Join(", ", row));  
}
```



.NET

Trino.Client

- How to use

Low level

```
ClientSession session = new TrinoConnectionProperties()
{
    Catalog = "tpch",
    Server = new Uri("https://trino.myhost.net/"),
    Auth = auth,
    SessionProperties = new Dictionary<string, string>()
    {
        { "dictionary_aggregation", "false" }
    }
}.GetSession();

RecordExecutor records = await RecordExecutor.Execute(
    logger: null, // optional detailed logging
    queryStatusNotifications: [(stats, error) =>
    {
        Console.WriteLine($"Processed rows:
{stats.processedRows}");
    }], // Output statistics to console as query runs
    session: session,
    statement: "select * from tpch.tiny.customer where custkey
= ?",
    queryParameters: new List<QueryParameter>() { new(751)
},
    bufferSize: 1024 * 1024 * 5, // 5 MB
    isQuery: true,
    CancellationToken.None).ConfigureAwait(false);
```

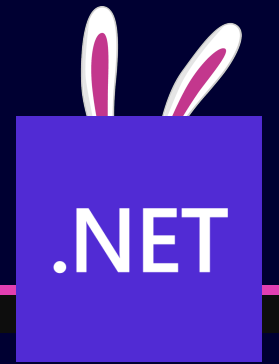
NET

Other features



- Auth modes: interface `ITrinoAuth`
 - In `Trino.Client`:
 - `TrinoJWTAuth`
 - `LDAPAuth`
 - In `Trino.Client.Auth`:
 - `TrinoOauthClientSecretAuth`
 - `TrinoAzureDefaultAuth`
 - Need more...
- `Trino.Client.Utils` extension methods
 - `public static DataTable BuildDataTable(this IList<TrinoColumn> columns)`
 - `public async static Task<DataTable> BuildDataTableAsync(this RecordExecutor recordExecutor)`

What's Next?



- Auth modes
- Spooling protocol support
 - Performance!
- Build/test/nuget.org distribution pipeline on Github
- Less restrictive type handling for ADO.NET?

Live on Github!



- [trinodb/trino-csharp-client](https://github.com/trinodb/trino-csharp-client): C# client for Trino
- Ready for PRs/reviewers!

