Exploring Mapepire, the new way to access Db2!

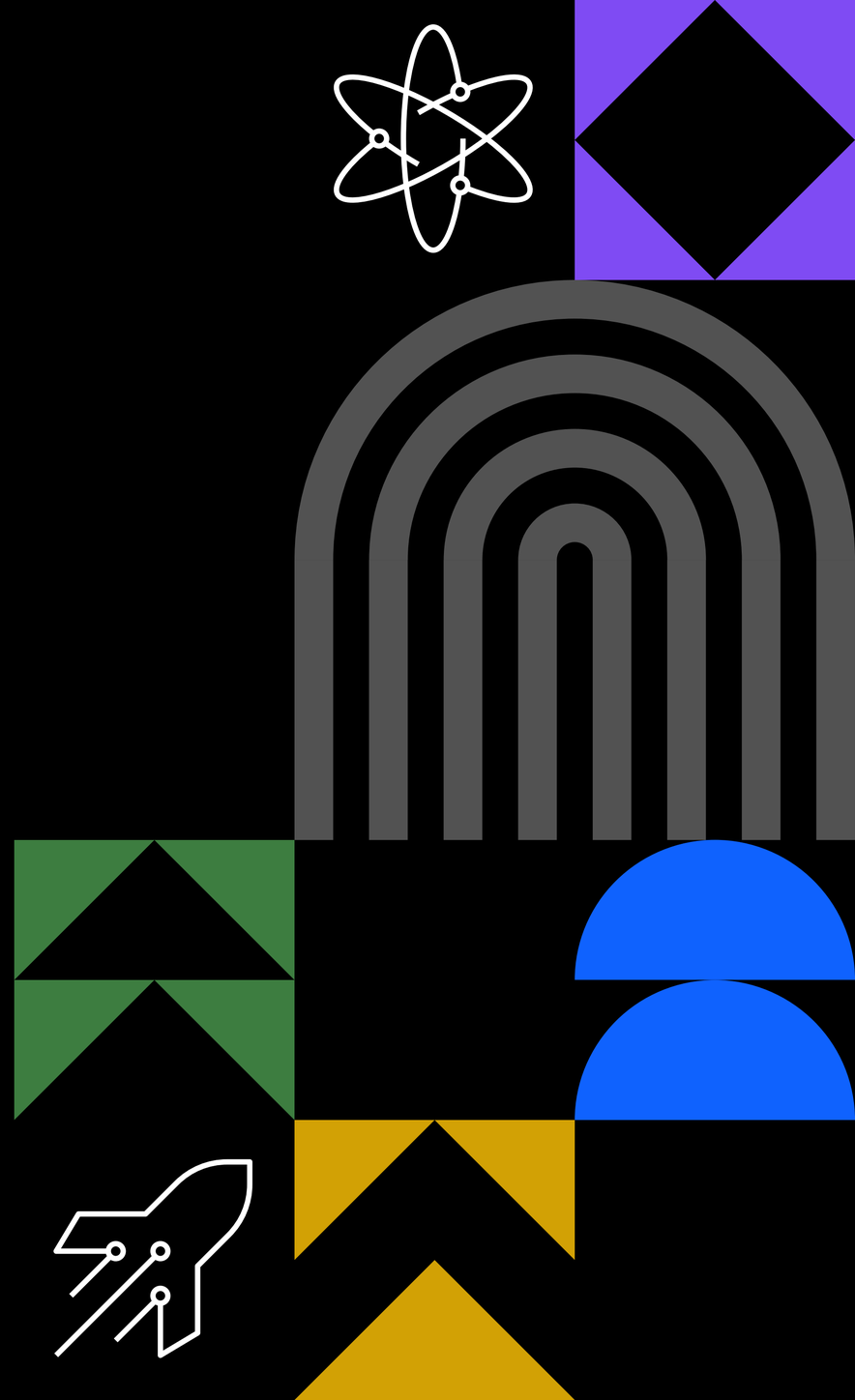
Jesse Gorzinski
Senior Business Architect, IBM

Agenda

01 What

02 Why

03 How



What is Mapepire? https://mapepire-ibmi.github.io/

Welcome to Mapepire

A cloud-friendly IBM i database access layer, built with simplicity and performance in-mind.

Find out more →

Pick your client language ⓘ

Why Mapepire?

Mapepire (pronounced ‘mapəpɪə’ or ‘MAH-pup-ee’) is a database access layer written on top of secure web sockets. It was built to make developing modern applications in .NET Core, Node.js, PHP, and the likes, easier when using Db2 for i.

Why the name?

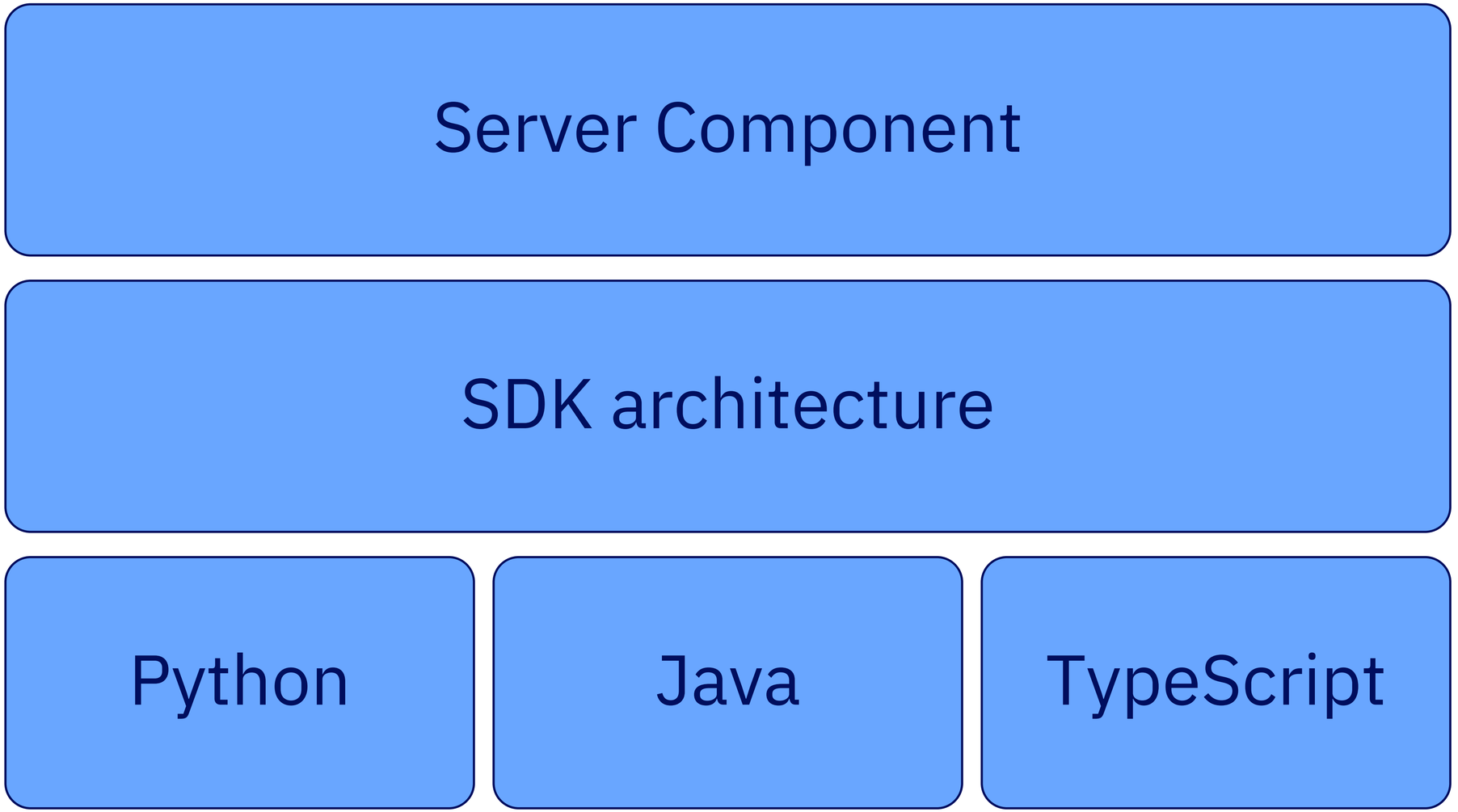
- Mapepire is a venomous pit viper from South America.
- Speed

J

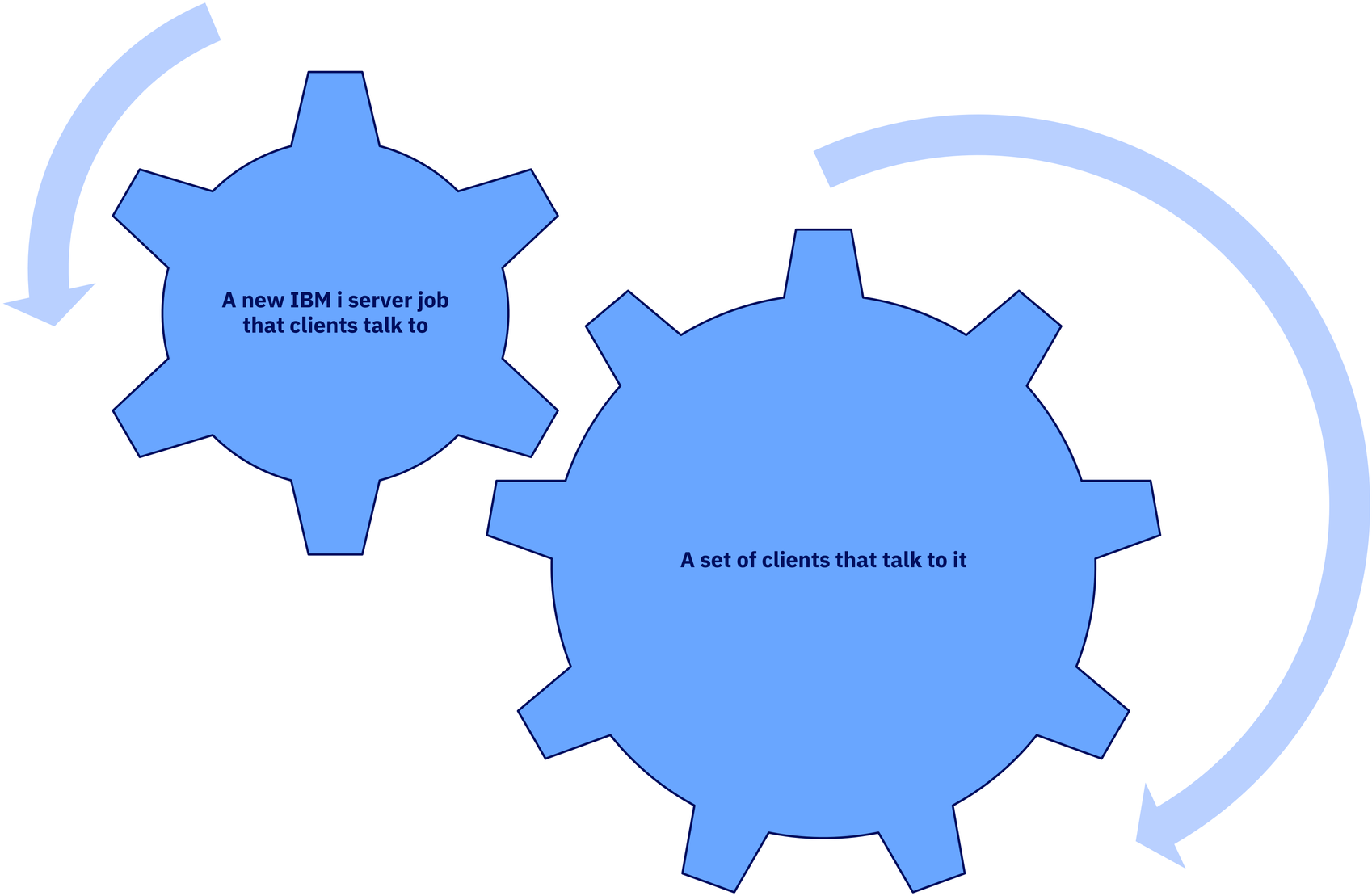
Mapepire Origin Story….

January 2020	February 2022	March 2022	July 2023	August 2024
• VSCode "Code for IBM i" extension includes basic Db2 support	• Work begins on Server component to power Db2 features in VSCode	• First release of VSCode Db2 for i extension	• VSCode Db2 for i extension publishes v0.3.0, the first release leveraging server component (v0.3.0)	• Mapepire is born!

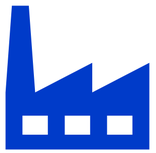
Mapepire Components



…but what IS IT?



Truth In Advertising



Already being used in production!

Technology Preview means Technology Preview

Agenda

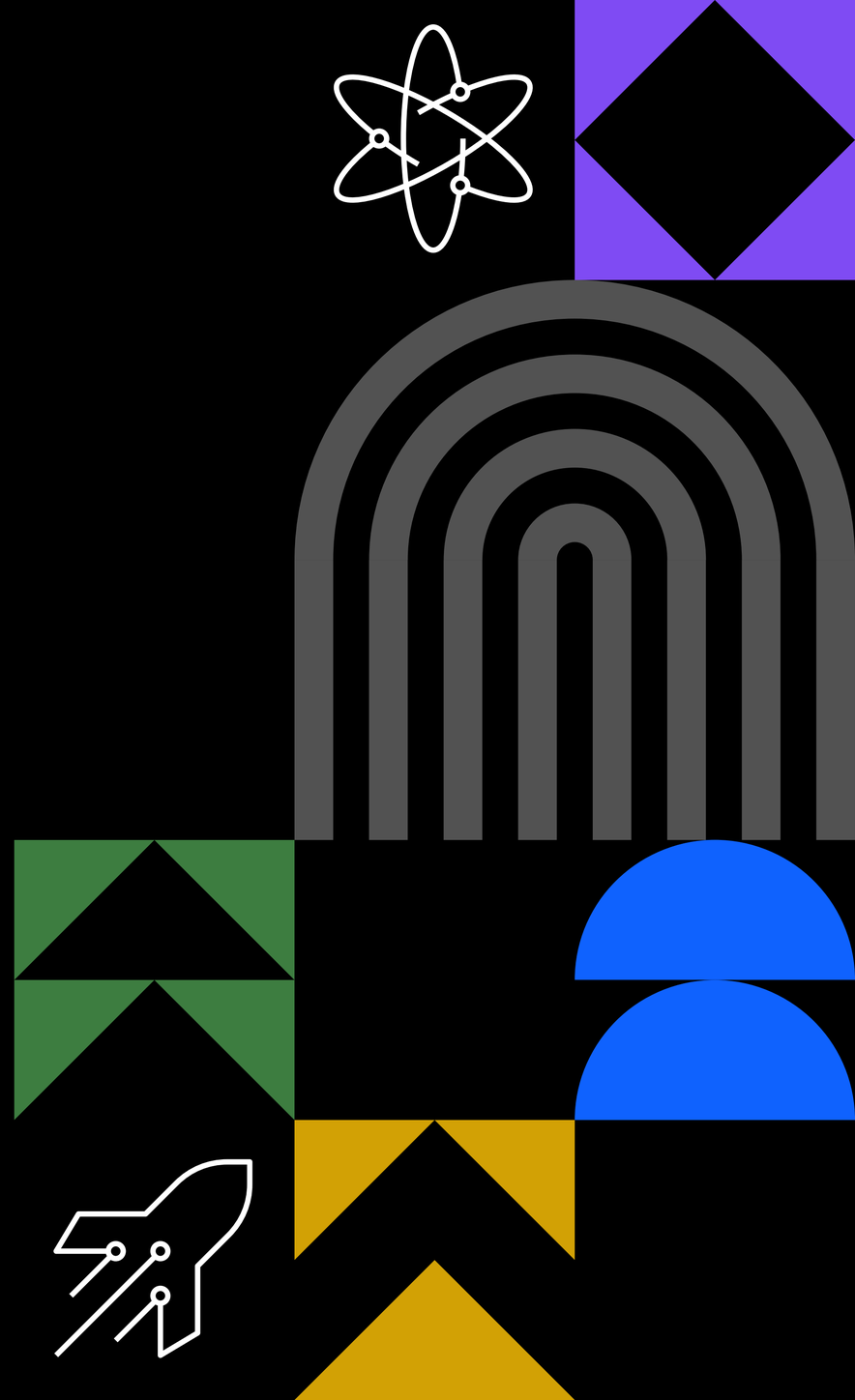
- What?
- Why?
- How?

Agenda

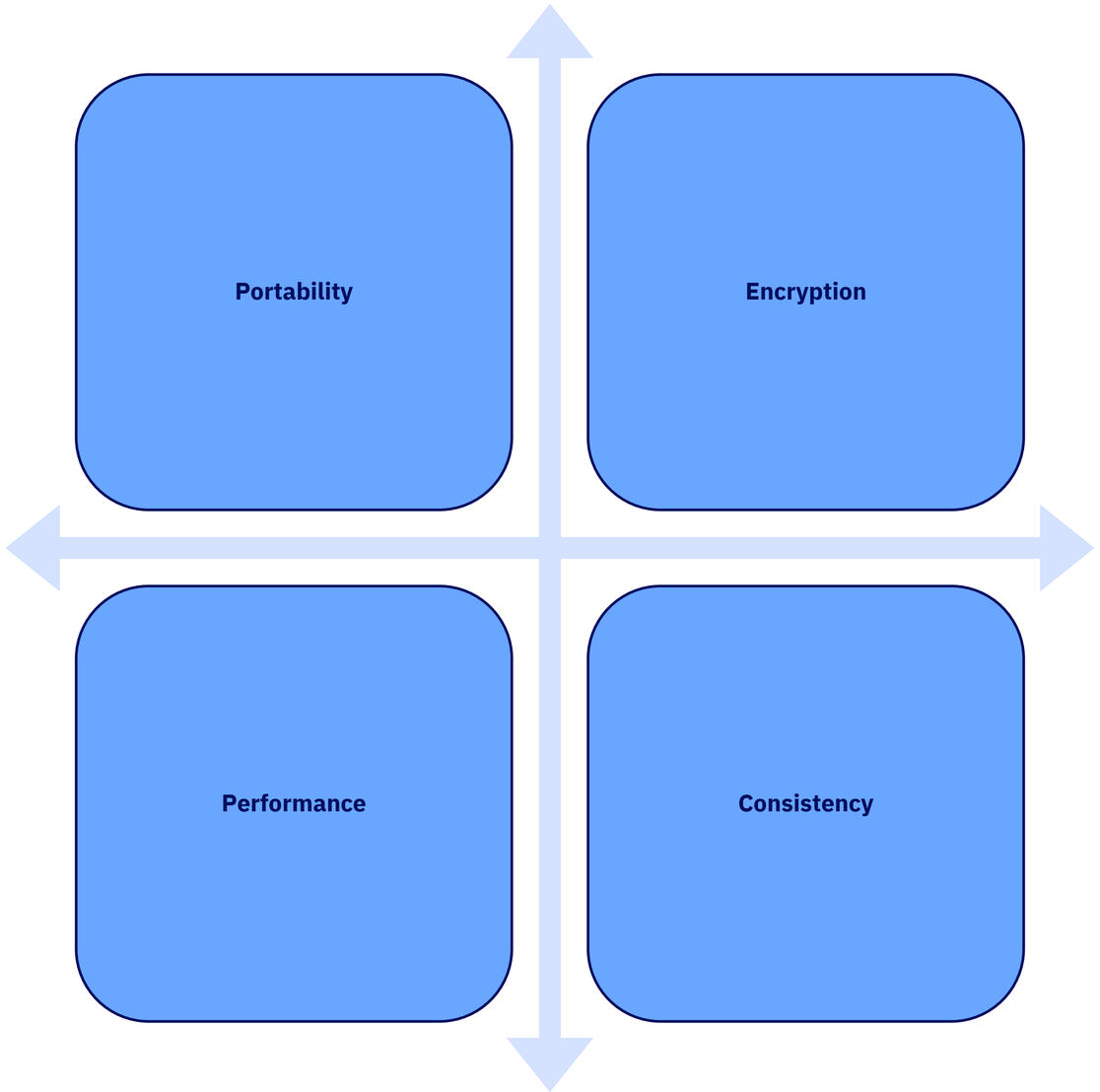
01 What

02 Why

03 How



Core tenets



Mapepire vs. JDBC and ODBC

	JDBC	ODBC	Mapepire
Needs only a single port			✅
Data is always encrypted			✅
Manageable via system exit points	✅	✅	✅
Enhanced CCSID support	✅		✅
Runs in WatsonX.ai Jupyter notebooks			✅
Runs in lightweight containers (for instance Alpine Linux)	✅		✅
Directly supports multiple client languages			✅

Single port? Big deal!

TCP distance to first database operation

JDBC/ODBC

Mapepire

Sample JDBC program

```
try (AS400 hi = new AS400("myhostname", "uid".toCharArray(), "password".toCharArray())) {
    AS400JDBCDataSource ds = new AS400JDBCDataSource(hi);
    Connection conn = ds.getConnection();
    Statement s = conn.createStatement();
    s.executeQuery("select * from QIWS.QCUSTCDT");
    ResultSet rs = s.getResultSet();
    while(rs.next()) {
        System.out.println(rs.getString(1));
    }
    System.out.println("done");
}
```

11 Distinct TCP Flows!!

- Host Server
  - 1::S - 7003 - Exchange Client/Server Attributes
  - 1::R - F003 - Exchange Client/Server Attributes Reply
  - 1::S - 7004 - Retrieve Signon Information
  - 1::R - F004 - Retrieve Signon Information Reply
  - 1::S - 7006 - End Job Request
  - 2::S - 7001 - Exchange Random Seeds
  - 2::R - F001 - Exchange Random Seeds Reply
  - 2::S - 7002 - Start Server
  - 2::R - F002 - Start Server Reply
  - 2::S - 1F80 - Set Attributes
  - 2::R - 2800 - SQL Requested Data Returned
  - 2::S - 1D00 - Create and init RPB with no based-on RPB
  - 2::S - 1803 - Prepare/Describe
  - 2::R - 2800 - SQL Requested Data Returned
  - 2::S - 180E - Open/Describe/Fetch
  - ⚠ 2::R - 2800 - SQL Requested Data Returned

How to encrypt data with _DBC

1. Log into DCM
2. Create a local certificate authority (CA) store
3. Create a local CA certificate
4. Record the value of the autocreated CA label
5. Create the *SYSTEM certificate store (if needed)
6. Create a new server certificate
7. Sign the server certificate with your local CA
8. Assign new server certificate to host server applications
9. Restart Host Servers

10. On client, download the server's certificate authority to a local truststore (or configure TLS to ignore completely)

How to encrypt data with Mapepire

Step 1
- Use Mapepire

Step 2
- Go to bed

What does TLS provide?

Encryption

- Data isn't sent "in the clear"

Authentication

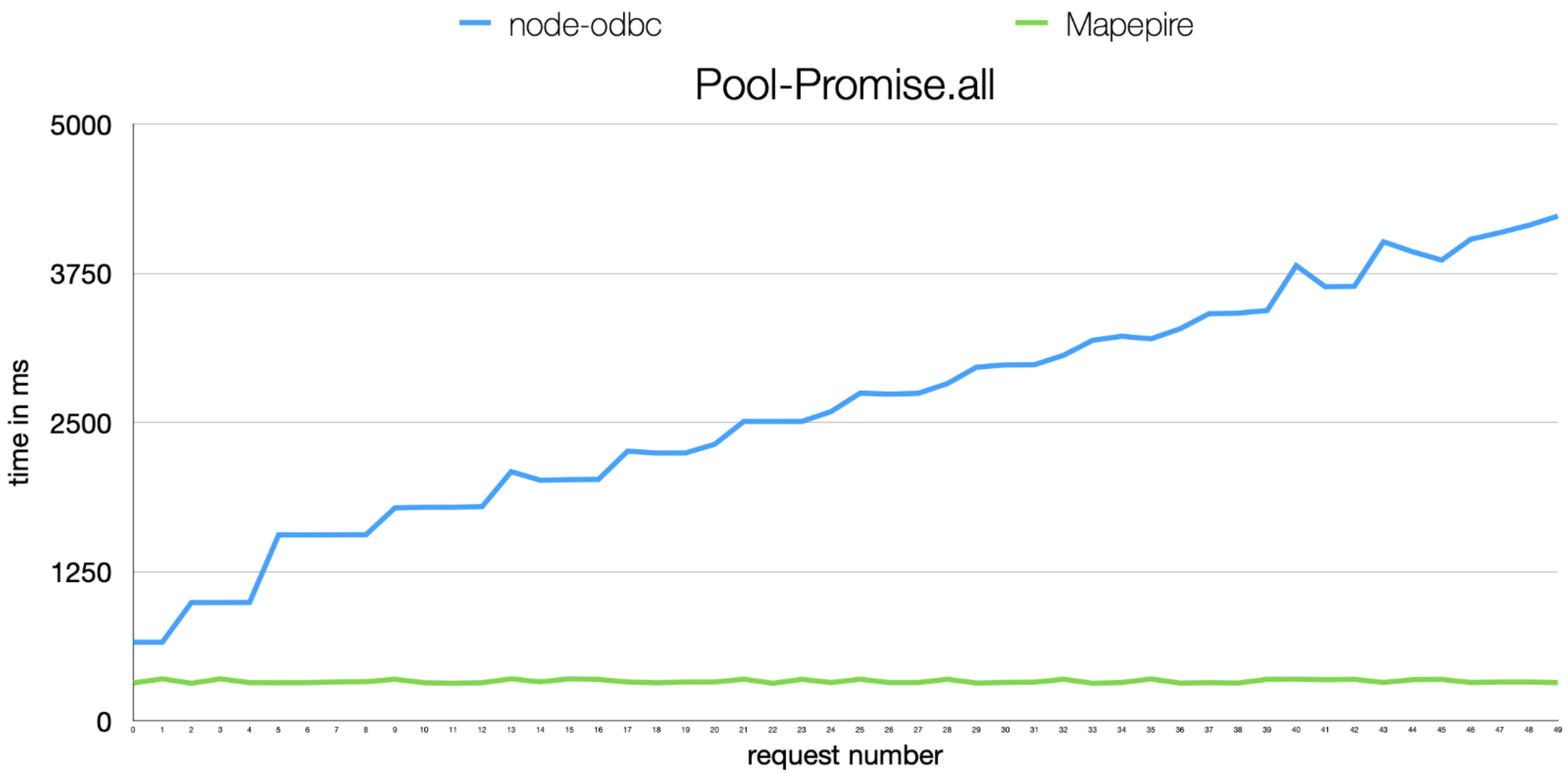
- Client ensures the server certificate is valid
- Client ensures the server certificate is signed by a trusted authority
- Client checks that the hostname matches that of the certificate

How does Mapepire make it so easy?

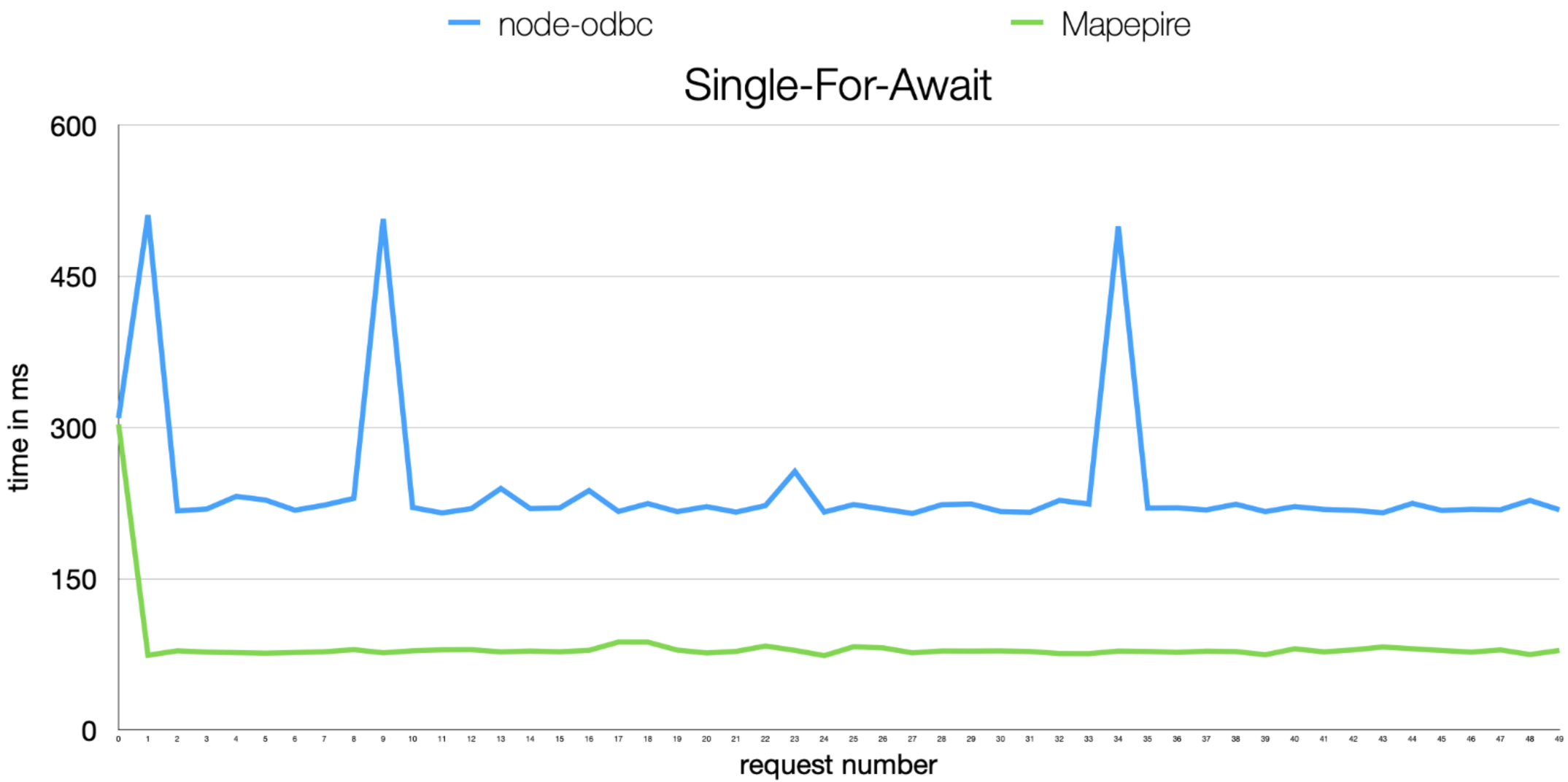
- Admin has explicitly configured a certificate
- DCM has database server configured for TLS [future enhancement]
- LetsEncrypt certificates are present

- Autogenerate a self-signed certificate

Some performance comparisons



Some performance comparisons



The biggest benefit of Mapepire….

The biggest benefit of Mapepire…. Portability!!

	JDBC	ODBC	Mapepire
Runs in WatsonX.ai Jupyter notebooks	×	×	√
Runs in Rocket AI Hub programmer portal	×	×	√
Runs in Rocket Cognitive Environment	√*	×	√
Runs in Alpine Linux containers	√	×	√
Runs in Raspberry Pi	√	×	√
Runs in Arduino	×	×	√

Is the biggest benefit also the biggest weakness?

Practices for ODBC/JDBC Management and Performance Tuning Still Apply

Mapepire's back-end is JDBC

Mapepire is an interface in front of JTOpen and JDBC
Mapepire utilizes QZDASOINIT or, more likely, QZDASSINIT (S = "secure") jobs
All considerations for ODBC/JDBC server job scalability and security still apply

Security

Mapepire runs "locally" on your IBM i, which helps to control ODBC/JDBC connection sprawl
Object authority still applies
Any ODBC/JDBC exit points will still work to control traffic and access

IBM | Documentation | Search in IBM i 7.5

IBM i

Change version

7.5

Show full table of contents

Filter on titles

- Database server
- Data queue server
- Network print server
- Central server
- Remote command and distributed program call server
- Signon server
- Examples: Exit programs
- Console Advanced Topics
- IBM i Access Client Solutions: Application Packages
- IBM Navigator for i

All products / IBM i / 7.5 /

Database server

Last Updated: 2024-10-07

Identify exit points for IBM i database serving.

The database server has five different exit points defined:

1. QIBM_QZDA_INIT
   - Called at server initiation
2. QIBM_QZDA_NDB1
   - Called for native database requests
3. QIBM_QZDA_SQL1
   - Called for SQL requests
4. QIBM_QZDA_SQL2
   - Called for SQL requests
5. QIBM_QZDA_ROI1
   - Called for retrieving object information requests and SQL catalog functions

Security BRAND NEW

Mapepire Configuration file lets you filter on users and IP addresses

```
# Allow anyone to connect
allow   * @ *

# Disallow any user profile starting with 'Q'
Deny   Q* @ *
```

Security BRAND NEW

Mapepire Configuration file lets you filter on users and IP addresses

```
# Deny by default
deny   * @ *

# Allow only specific users, and only from 192.168.*.*
allow  appusr1 @ 192.168.*
allow  appusr2 @ 192.168.*
```

How to manage the JDBC workload?

By default, all QZDASOINIT/ QZDASSINIT jobs run in QUSRWRK

Use SET_SERVER_SBS_ROUTING to apply custom memory pools or capping groups

Questions:

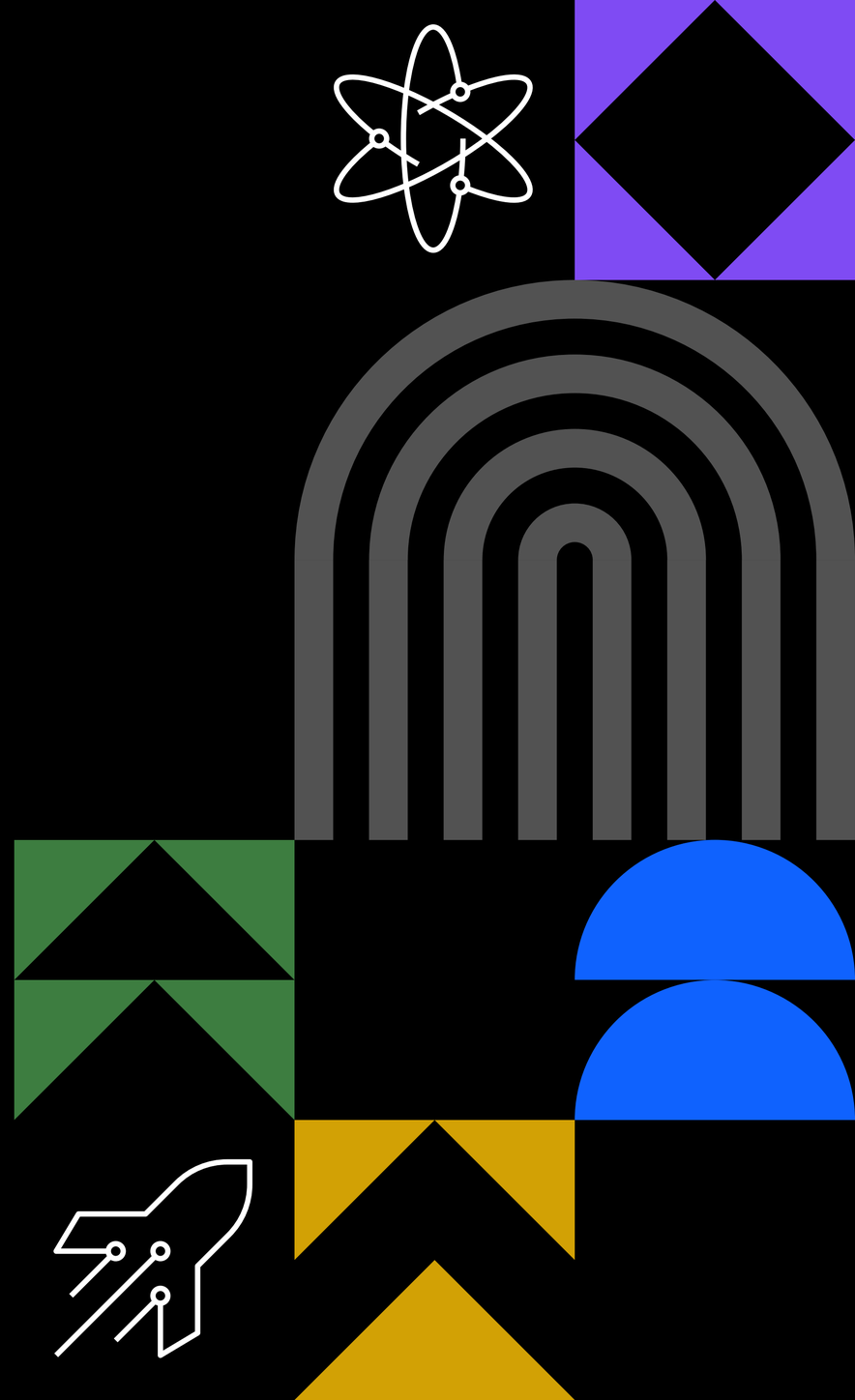
- How to control out-of-control queries from query tools?
- How to know which application is using up resources?
- How to manage your JDBC jobs more effectively?
- How to let critical users get the resources they need while not letting long queries take over the system?

Agenda

01 What

02 Why

03 How



Install the Mapepire server

Installation

Option 1: RPM (recommended)

```
yum install mapepire-server
```

(if you need help getting started with RPMs, please see http://ibm.biz/ibmi-rpms)

Option 2: manual installation

1. Make a “download” directory on IBM i by running the following from an SSH terminal:

```
mkdir -p /opt/download
```

2. Download the distribution zip file (filename will look something like `mapepire-server-v___.zip`) from the release page and save it to the download directory you created. Rename the file to

Launch the Mapepire server

- Install the Mapepire server component `yum install mapepire-server`
- Install Service Commander `yum install service-commander`
- Launch mapepire `sc start mapepire`

```
-bash-5.2$ sc start mapepire
Performing operation 'START' on service 'mapepire'
Service 'Mapepire Server' successfully started
```

JavaScript client

- Install the mapepire client with npm install @ibm/mapepire-js
- Copy the .env.sample file to a file named .env and fill in the required values

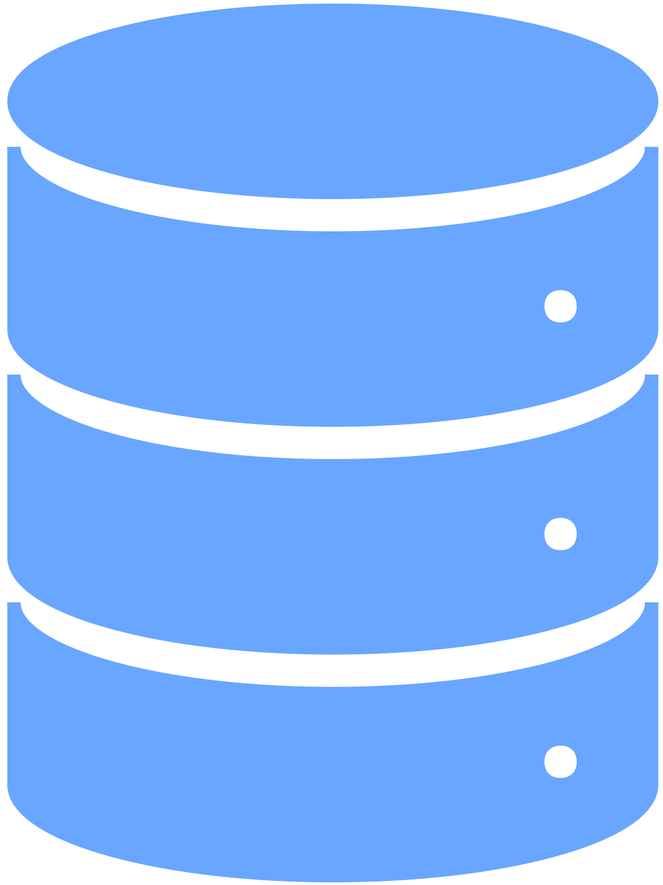
.env

```
# defaults to localhost
VITE_SERVER=mymachine.somelab.com

# defaults to 8076
VITE_PORT=8076

VITE_DB_USER=jonsmith
VITE_DB_PASS=letmein
```

Now it’s time to write some queries!

We have the following functionality available to us

- Connect to the database
- Run CL commands
- Run SQL commands
- Run SQL commands as a batch
- Get server job
- Get Mapepire version
- Check liveliness
- Set config
- Get trace data
- Close a connection



Connecting to the database

- Use the connect method of the SQLJob class

```
// Connect to the database
const creds = ENV_CREDS ;
const job = new mapepire.SQLJob();
await job.connect(creds);
```

Executing a query

Use the query method of SQLJob to construct a query
Then call the queries execute method

```
await job.connect(creds);
const query = await job.query<any>("select * from sample.department");
const res = await query.execute();
await query.close()
await job.close();
```

Executing a prepared statement

Use the query method of SQLJob to construct a query
Add the parameters for the prepared query
Then call the queries execute method

```
const job = new SQLJob();
await job.connect(creds);
const query = await job.query<any>(
  "SELECT * FROM SAMPLE.SYSCOLUMNS WHERE COLUMN_NAME = ?",
  {
    parameters: ["Value"],
  }
);
const res = await query.execute();
await query.close();
await job.close();
```

Executing a batch of statements

Use the query method of SQLJob to construct a prepared statement
Use a 2d array of parameters to ensure the query is constructed as a batch operation
Then call the queries execute method

```
let query = job.query("INSERT INTO SAMPLE.EMPLOYEES values (?, ?)", {
  parameters: [
    ["SANJULA", "416 345 0879"],
    ["TONGKUN", "647 345 0879"],
    ["KATHERINE", "905 345 1879"],
    ["IRFAN", "647 345 0879"]
  ],
});
let res = await query.execute();
```

Executing statements using a pool

Initialize a new pool object
Call the pool's execute method to run a query using any free job
Call the pool's end method to destroy the pool and cleanup resources

```
let pool = new Pool({ creds, maxSize: 5, startingSize: 5 });
await pool.init();
// Initiate a bunch of jobs
const executedPromises = [
  pool.execute("select * FROM SAMPLE.SYSCOLUMNS"),
  pool.execute("select * FROM SAMPLE.SYSCOLUMNS"),
  pool.execute("select * FROM SAMPLE.SYSCOLUMNS"),
];
const res = await Promise.all(executedPromises);
await pool.end();
```

SQLJob	PoolJob
Queries are run in a single job	Queries are run in possibly-reused jobs, and possibly multiple jobs
Handles its own connection	Retrieves connections from a shared pool
Slower due to repeated connection overhead	Faster due to connection re-use

S

Consistent SDK behavior across languages

Guided by a unified reference architecture

Similar experiences

- Class names
- Method names
- Throwable types
- Input parameters
- Configuration options

PowerWire article from Andy Youens

https://powerwire.uk/mapepire-introduction/

What’s coming…

…. in 2026?

- Kerberos
- Improved LOB capabilities
- PHP client
- Full JDBC client
- Thorough API documentation
- Delivery as part of IBM i OS?

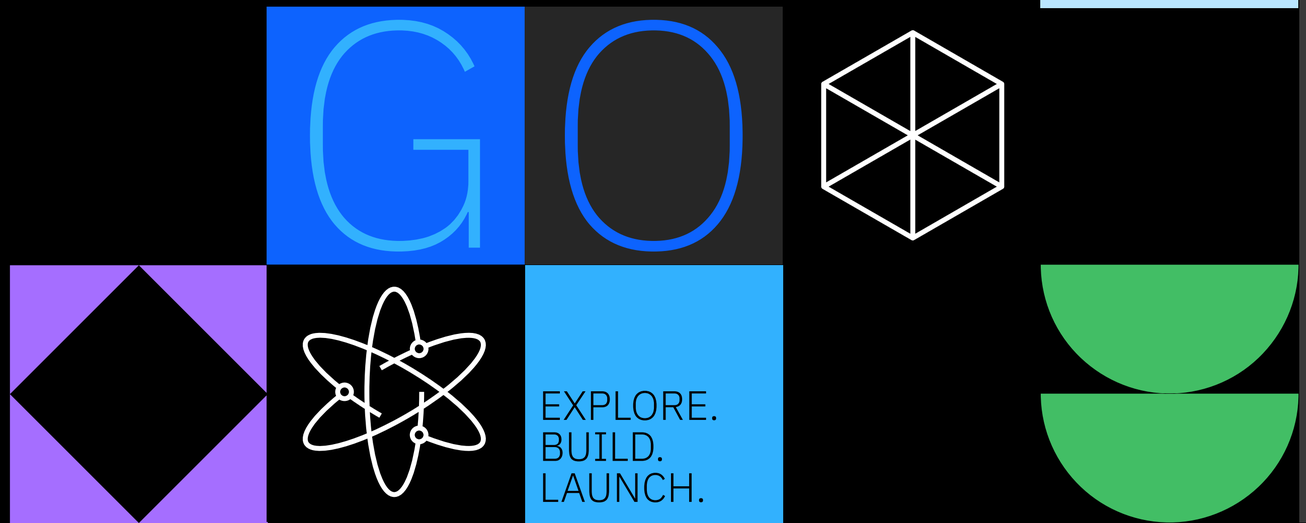
…. At some point?

- C# client
- Improved Go client
- Data compression

Takeaways

Config	Suitable performance
	Always encrypted
Flexibility	Any Hardware
	Any Language

Thank you



Firstname Lastname

Company, Job Title

Contact Info

Notices and disclaimers

This document is distributed “as is” without any warranty, either express or implied. In no event shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.

Case studies and client examples are presented as illustrations of how customers or IBM has used IBM products in production or test environments and the results they may have observed. Actual performance, cost, savings or other results in other operating environments may vary.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM.

Not all offerings are available in every country in which IBM operates.

Any statements regarding IBM’s future direction, intent or product plans are subject to change or withdrawal without notice.

Certain comments made in this presentation may be characterized as forward looking under the Private Securities Litigation Reform Act of 1995.

Forward-looking statements are based on the company’s current assumptions regarding future business and financial performance. Those statements by their nature address matters that are uncertain to different degrees and involve a number of factors that could cause actual results to differ materially. Additional information concerning these factors is contained in the Company’s filings with the SEC.

Copies are available from the SEC, from the IBM website, or from IBM Investor Relations.

Any forward-looking statement made during this presentation speaks only as of the date on which it is made. The company assumes no obligation to update or revise any forward-looking statements except as required by law; these charts and the associated remarks and comments are integrally related and are intended to be presented and understood together.