Forget ODBC! Here's a New Db2 Connector

Jonathan Zak, IBM
Software Developer
jonathan.zak@ibm.com

Sanjula Ganepola, IBM
Software Developer
sanjula.ganepola@ibm.com

Toronto 2024
NAViGATE

COMMON
LEARN. ENGAGE. GROW.

Agenda

- Mapepire Overview

- Architecture and Core Tenets

- Comparisons versus JDBC and ODBC

- Deep dive into Node.js client SDK

- Demo

What is Mapepire?

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 ⓘ

Super easy to use way to access Db2 for i from any application

Java   node JS   python

C#   php

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

Server Component

SDK architecture

| Python | Java | TypeScript | C# | PHP | FUTURE |

How does it work?

Connect to Database

Communication over
secure web sockets

{

    "id": "id2",
    "type": "connect",
    "technique": "tcp",
    "application": "Java client"

}

IBM i

Application

Server Component

Application uses
Mapepire client SDK

{

    "id": "id2",
    "job": "112480/QUSER/QZDASOINIT",
    "success": true

}

Managed using
Service Commander

How does it work?

## Query the Database

Request

```json
{
    "id": "query3",
    "type": "sql",
    "sql": "SELECT DEPTNO FROM SAMPLE.DEPARTMENT
WHERE DEPTNO = 'A00'",
    "terse": false,
    "rows": 100
}
```

Response

```json
{
    "id": "query3",
    "has_results": true,
    "update_count": -1,
    "metadata": {
        "column_count": 1,
        "job": "112480/QUSER/QZDASOINIT",
        "columns": [
            {
                "name": "DEPTNO",
                "type": "CHAR",
                "display_size": 3,
                "label": "DEPTNO",
                "precision": 3,
                "scale": 0
            }
        ]
    },
    "data": [
        {
            "DEPTNO": "A00"
        }
    ],
    "is_done": true,
    "success": true
}
```
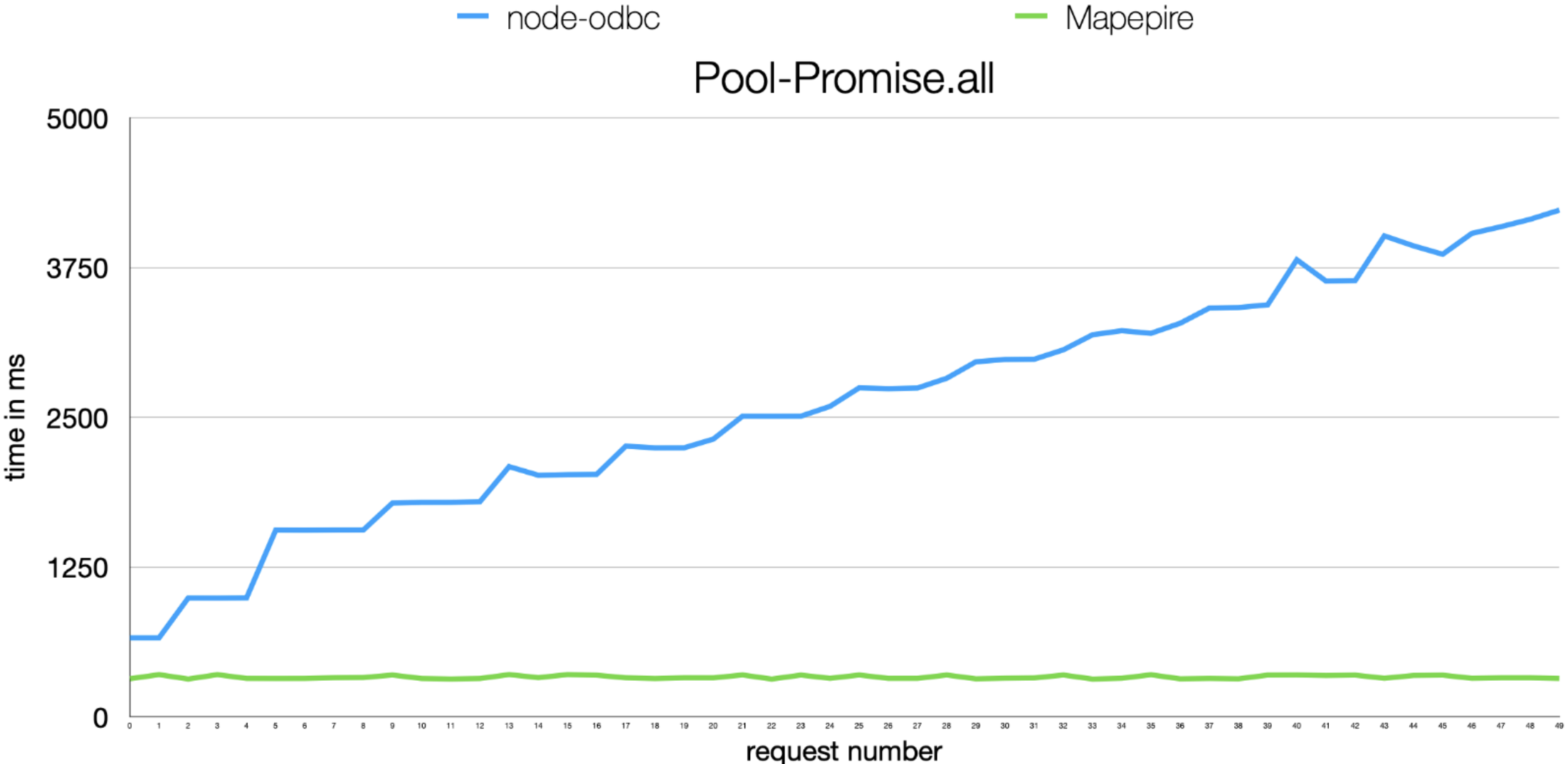
Core Tenets
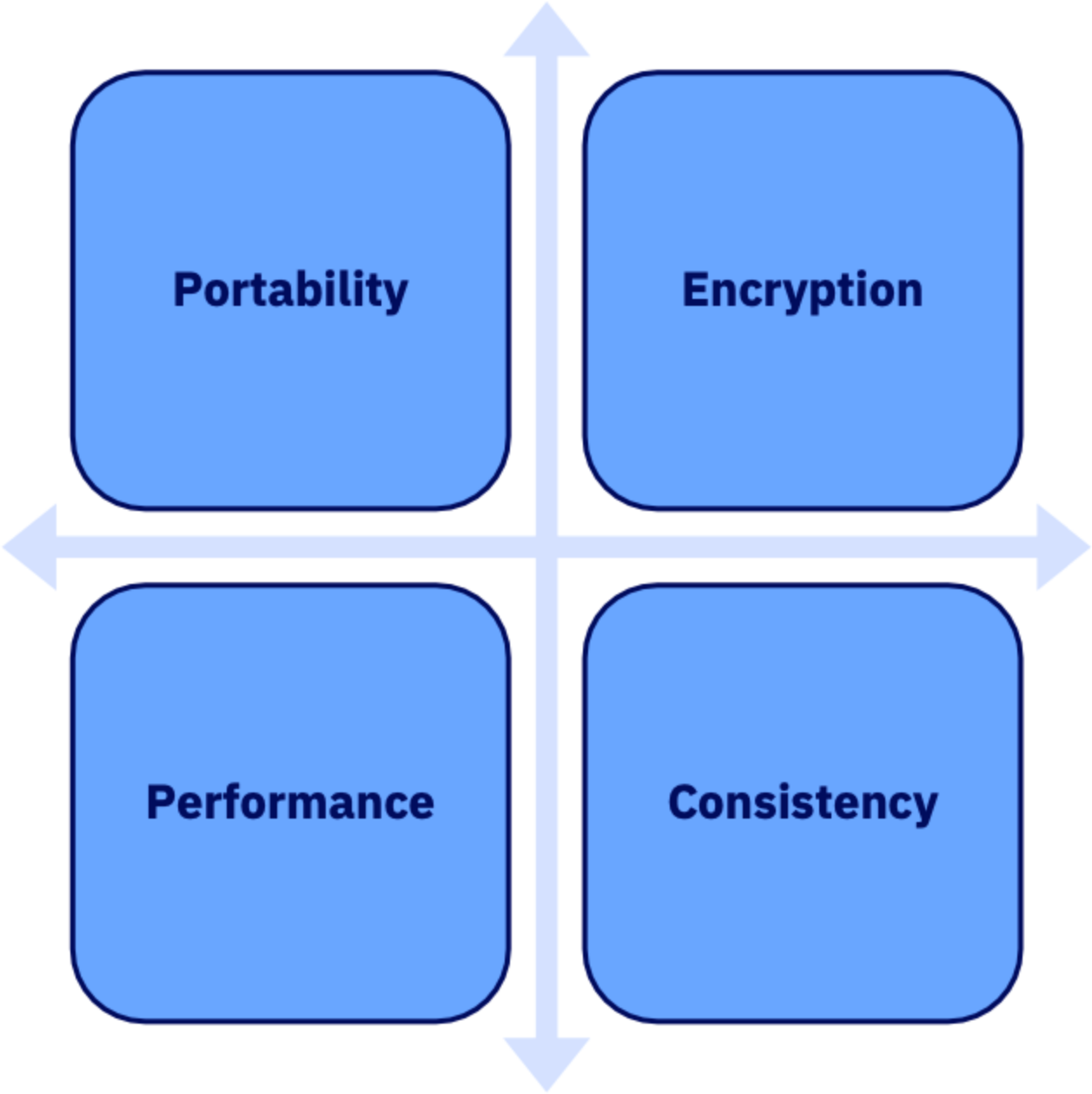
The biggest benefit of Mapepire…. Portability!!

	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	✗	✗	✔

Some performance comparisons

Pool-Promise.all — performance comparison chart showing node-odbc (blue) and Mapepire (green) lines, with time in ms on the y-axis (0 to 5000) and request number on the x-axis (0 to 49).

Some performance comparisons

Single-For-Await (node-odbc vs Mapepire); x-axis: request number; y-axis: time in ms
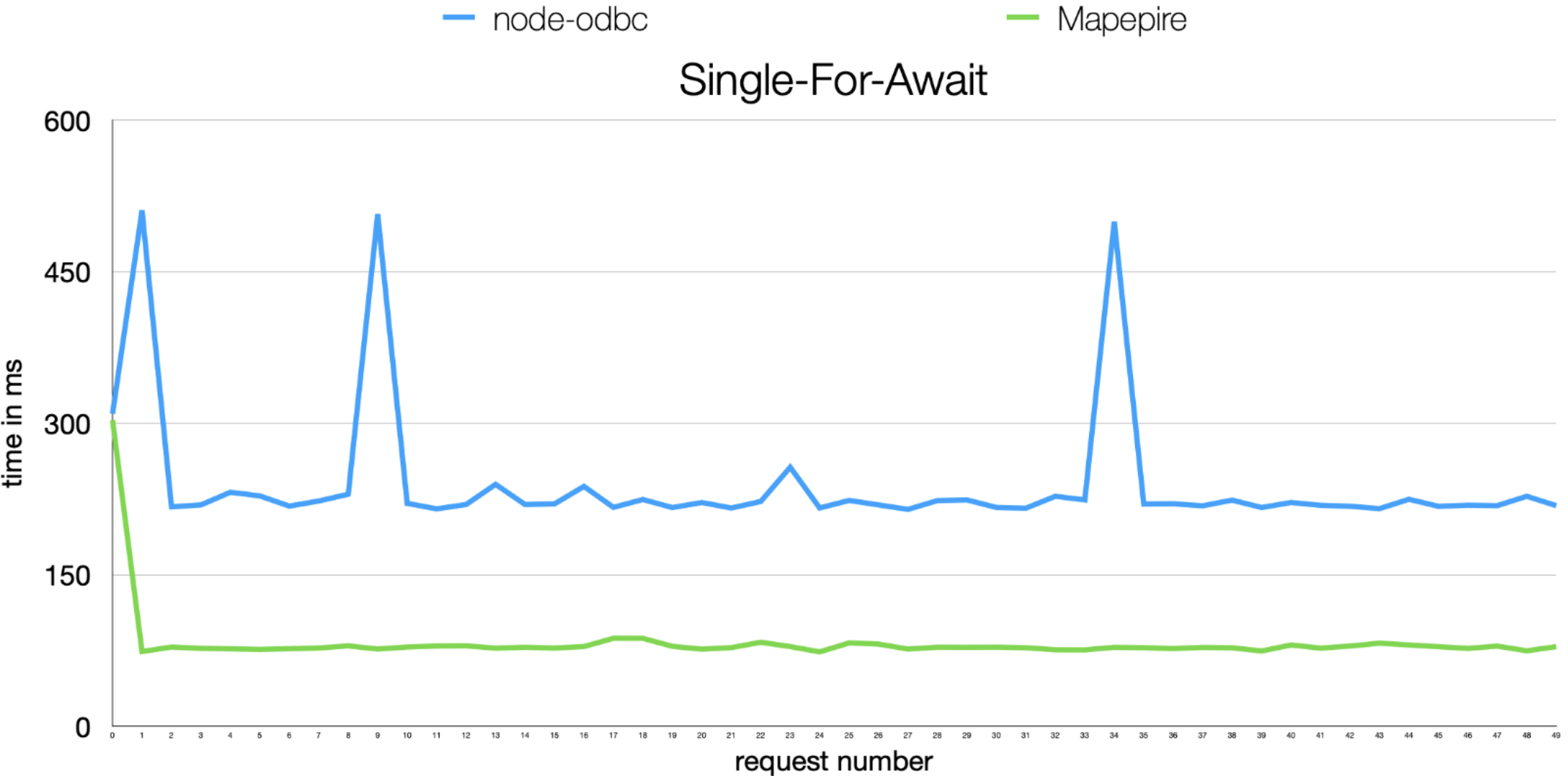
How to encrypt data with JDBC/ODBC

1. Log into DCM
2. Create a local certificate authority (CA) store
3. Create a local CA certificate
4. Record the value of the auto created 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 trust store (or configure TLS to ignore completely)

How to encrypt data with Mapepire?

Step 1 { •Use Mapepire

Step 2 { •Go to bed

How does Mapepire make it so easy?

Option 1: Custom certificate

- Admin explicitly defined a custom certificate by configuring a certificate store:
  - File name: `/QOpenSys/etc/mapepire/cert/server.jks`
  - Format: `JKS`
  - Store Password: `mapepire`
  - Key Password: `mapepire`
  - Certificate Alias: `mapepire`
- Check out documentation for full instructions: https://mapepire-ibmi.github.io/guides/sysadmin/

Option 2: Let's Encrypt

- Use Let's Encrypt (ex. generated by CertBot)
- Mapepire server will automatically use it as the server certificate
- Certificate must exist in the following location used by CertBot:
  `/etc/letsencrypt/live/<hostname>`

Option 3: Self-signed certificate

- If no certificate, the server automatically generates its own self-signed certificate

14

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

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
  - Object authority still applies
  - Any ODBC/JDBC exit points will still work to control traffic and access

How to manage the JDBC workload?

- By default, all QZDASOINIT/ QZDASSINIT jobs run in QUSRWRK

- Questions:
  - How to control out-of-control queries from query tools?

  - How to know which application is using up resources?

  - How to let critical users get the resources they need while not letting long queries take over the system?

  - How to manage your JDBC jobs more effectively?

Separate jobs by application, user, etc.

- Configure your QZDASSINIT jobs to run in separate subsystems, based on your criteria
  - JDBCSHOP
  - JDBCADHOC
  - JDBCNODE

- Then performance "waits" can be aggregated by subsystem and you can configure memory, etc. per subsystem

- Easier troubleshooting as JDBC jobs from different applications will not interact

- Also limit with these techniques https://www.ibm.com/support/pages/setting-limitations-resources-used-qzdasoinit-prestart-jobs

- Details: https://www.seidengroup.com/2022/05/04/simplify-with-subsystems/

Configure JDBC Prestart jobs

- Configure the right number of prestart jobs
  - ODBC/JDBC prestart jobs are QZDASO(S)INIT in QUSRWRK

- Check out your current configuration:
  - DSPSBSD SBSD(QUSRWRK)
  - Choose 10, Prestart job entries
  - Type 5 next to QZDASOINIT

```
                        Display Prestart Job Entry Detail
                                                    System:     SV12
Subsystem description:     QUSRWRK           Status:    ACTIVE

Program  . . . . . . . . . . . . . . . . . . . . :     QZDASOINIT
  Library  . . . . . . . . . . . . . . . . . . . :       QSYS
User profile . . . . . . . . . . . . . . . . . . :     QUSER
Job  . . . . . . . . . . . . . . . . . . . . . . :     QZDASOINIT
Job description  . . . . . . . . . . . . . . . . :     QDFTSVR
  Library  . . . . . . . . . . . . . . . . . . . :       QGPL
Start jobs . . . . . . . . . . . . . . . . . . . :     *YES
Initial number of jobs . . . . . . . . . . . . . :     1
Threshold  . . . . . . . . . . . . . . . . . . . :     1
Additional number of jobs  . . . . . . . . . . . :     2
Maximum number of jobs . . . . . . . . . . . . . :     *NOMAX
Maximum number of uses . . . . . . . . . . . . . :     200
Wait for job . . . . . . . . . . . . . . . . . . :     *YES
Pool identifier  . . . . . . . . . . . . . . . . :     1
```

Default ODBC/JDBC prestart job settings

Low defaults
- Initial jobs = 1, threshold = 1, additional jobs = 2

- Change as needed:
  **CHGPJE SBSD(QSYS/QUSRWRK) PGM(QSYS/QZDASOINIT)
  STRJOBS(*YES) INLJOBS(xx) THRESHOLD(xx)
  ADLJOBS(xx)**

- How to determine **optimal values**? DSPACTPJ (coming up)

```
                       Display Prestart Job Entry Detail
                                                    System:      SV12
Subsystem description:    QUSRWRK         Status:    ACTIVE

Program . . . . . . . . . . . . . . . . . . . . . :   QZDASOINIT
   Library . . . . . . . . . . . . . . . . . . . :      QSYS
User profile . . . . . . . . . . . . . . . . . . :   QUSER
Job . . . . . . . . . . . . . . . . . . . . . . . :   QZDASOINIT
Job description . . . . . . . . . . . . . . . . . :   QDFTSVR
   Library . . . . . . . . . . . . . . . . . . . :      QGPL
Start jobs . . . . . . . . . . . . . . . . . . . :   *YES
Initial number of jobs . . . . . . . . . . . . . :   1
Threshold . . . . . . . . . . . . . . . . . . . . :   1
Additional number of jobs . . . . . . . . . . . . :   2
Maximum number of jobs . . . . . . . . . . . . . :   *NOMAX
Maximum number of uses . . . . . . . . . . . . . :   200
Wait for job . . . . . . . . . . . . . . . . . . :   *YES
Pool identifier . . . . . . . . . . . . . . . . . :   1
```

How many jobs are needed?

- DSPACTPJ SBS(QUSRWRK) PGM(QZDASOINIT)

- More details: https://www.ibm.com/docs/en/i/7.4?topic=jobs-tuning-prestart-job-entries

```
_Prestart jobs:
   Current number  . . . . . . . . . . . . . . . . . :      29
   Average number  . . . . . . . . . . . . . . . . :      21.9
   Peak number  . . . . . . . . . . . . . . . . . :      49

 Prestart jobs in use:
   Current number  . . . . . . . . . . . . . . . . . :      27
   Average number  . . . . . . . . . . . . . . . . :      18.4
   Peak number  . . . . . . . . . . . . . . . . . :      46
```

```
_Program start requests:
   Current number waiting  . . . . . . . . . . . . :       0
   Average number waiting  . . . . . . . . . . . . :       .0
   Peak number waiting  . . . . . . . . . . . . . :       6
   Average wait time  . . . . . . . . . . . . . . :       00:00:00.0
   Number accepted  . . . . . . . . . . . . . . . . :      100299
   Number rejected  . . . . . . . . . . . . . . . . :       0
```

JDBC vs ODBC vs Mapepire

	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

```
Port Mapper  →  Signon Server  →  Database Server
```

Mapepire

```
Mapepire
```
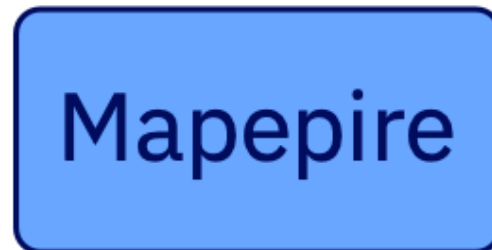
Distinct TCP flows for a JDBC program!!

```java
1   try (AS400 hi = new AS400("myhostname", "uid".toCharArray(), "password".toCharArray())) {
2       AS400JDBCDataSource ds = new AS400JDBCDataSource(hi);
3       Connection conn = ds.getConnection();
4       Statement s = conn.createStatement();
5       s.executeQuery("select * from QIWS.QCUSTCDT");
6
7       ResultSet rs = s.getResultSet();
8       while (rs.next()) {
9           System.out.println(rs.getString(1));
10      }
11      System.out.println("done");
12  }
```

```
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
```

Meanwhile....

✨ Mapepire only needs 2 TCP flows
1. Connect and allocate a job
2. Run a query

24

Installation made easy!

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 `mapepire-server-dist.zip`

If you have `wget` installed and have internet access, you could use wget to download the file from IBM i, for instance (replace with proper version):

How to get started with Mapepire?

- Install the mapepire client with npm install @ibm/mapepire-js
- Create a .env file specifying HOST, USER, PASSWORD and PORT (this is the Mapepire server port)

```
⚙ .env
1      HOST="myhost.somewhere.com"
2      PORT="8076"
3      USER="jimbob"
4      PASSWORD="letmein4"
5
```

```
const creds = {
    host: process.env.HOST,
    port: process.env.PORT,
    user: process.env.USER,
    password: process.env.PASSWORD,
};

const ca = await mapepire.getCertificate(creds);
creds.ca = ca.raw;
const job = new mapepire.SQLJob();
await job.connect(creds);
```

26

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
```

Now it's time to write some queries!

Here is some of the functionality we have available to us

- Connect to the database

- Run CL commands

- Run SQL commands

- Run SQL commands as a batch

- Run procedures

- Explain a statement

- Get server job

- Get Mapepire version

- Check liveliness

- Set config

- Get trace data

- Close a connection

Connecting to the database
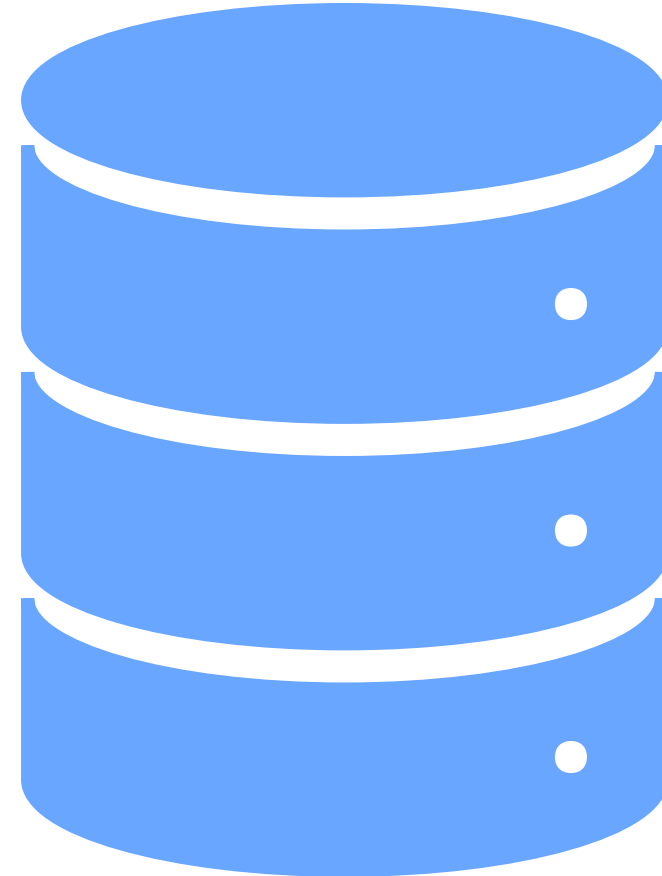
- Use the connect method of the
  SQLJob class

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

Get the server certificate

- Use the getCertificate method to retrieve the TLS certificate from the db2 server. If the server is using a self signed certificate, you can add it to your connection credentials to specify you trust this certificate.

```
const creds = ENV_CREDS ;
const ca = await mapepire.getCertificate(creds);
creds.ca = ca.raw;
```

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 procedure

- You can execute a procedure the same way that you would execute an SQL statement!

```
const job = new SQLJob();
await job.connect(creds);

const testProc = `
  create or replace procedure ${TEST_SCHEMA}.procedure_test_char(
    in p1 char(5),
    inout p2 char(6),
    out p3 char(7)
  )
  BEGIN
    set p3 = rtrim(p1) concat rtrim(p2);
    set p2 = '';
  END
`;

const queryA = job.query<any[]>(testProc);
await queryA.execute();
await queryA.close();
```

Executing a CL command

- Use the CL method of the SQLJob class to execute CL commands

```
const job = new SQLJob();
await job.connect(creds);
const query = await job.clcommand("WRKACTJOB");
const res = await query.execute();
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();
```

34

Explain an SQL statement

- Use the explain method of SQLJob to explain an sql statement and return the result

```
const job = new mapepire.SQLJob();
await job.explain("select * from SAMPLE");
await job.close()
```

Get Mapepire server version

- Use the getVersion method of SQLJob to retrieve the Mapepire server version

```
const job = new mapepire.SQLJob();
await job.getVersion()
```

Retrieve the count of pending transactions

- Use the getPendingTransactions method of SQLJob to retrieve the count of pending transactions

```
const job = new mapepire.SQLJob();
await job.getPendingTransactions()
```

Retrieve the number of ongoing requests for a job

- Use the getRunningCount method of SQLJob to retrieve the number of ongoing requests for the job

```
const job = new mapepire.SQLJob();
await job.getRunningCount()
```

Retrieve the status for the job

- Use the getStatus method of SQLJob to retrieve the status of the job
- Can be any of notStarted, connecting, ready, busy, ended

```
const job = new mapepire.SQLJob();
await job.getStatus();
```

Retrieve the trace file path

- Use the getTraceFilePath method of SQLJob to retrieve the file path of the trace file

```
const job = new mapepire.SQLJob();
await job.getTraceFilePath()
```

Get the server trace data

- Use the getTraceData method of SQLJob to retrieve the trace data from the backend

```
const job = new mapepire.SQLJob();
await job.getTraceData()
```

Get the SQLJob that is running the query

- Use the getHostJob method of Query class to return the SQLJob that is running the query

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

const queryJob = query.getHostJob();
```

Get the state of the query

- Use the getState method of Query class to return the state of the query
- Can be one of "NOT_YET_RUN", "RUN_MORE_DATA_AVAILABLE", "RUN_DONE", "ERROR"

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

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

```javascript
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

```javascript
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();
```

Set the backend log level

- Use the setTraceConfig method of the SQLJob class to set the trace level and destination for the Mapepire server
- In this case we are sending the trace data to a file, and we are using the most verbose level of logging

```
const job = new mapepire.SQLJob();
await job.setTraceConfig("FILE", "DATASTREAM")
```

Consistent SDK behavior access languages

- Guided by a unified reference architecture
  - https://mapepire-ibmi.github.io/reference/maintenance/referencearchitecture/

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

Node.js vs Java Implementation

Node.js

```javascript
// Initialize credentials
const creds: DaemonServer = { host: "HOST", port: 8076, user:
"USER", password: "PASSWORD", rejectUnauthorized: true, ca:
"CA" }


// Establish connection
const job = new SQLJob();
await job.connect(creds);



// Initialize and execute query
const query = job.query("SELECT * FROM SAMPLE.DEPARTMENT");
const result = await query.execute(3);



// Convert to JSON string and output
console.log(JSON.stringify(result));
```

Java

```java
// Initialize credentials
DaemonServer creds = new DaemonServer("HOST", 8085, "USER",
"PASSWORD", true, "CA");


// Establish connection
SqlJob job = new SqlJob();
job.connect(creds).get();



// Initialize and execute query
Query query = job.query("SELECT * FROM SAMPLE.DEPARTMENT");
QueryResult<Object> result = query.execute(3).get();



// Convert to JSON string and output
ObjectMapper mapper = new ObjectMapper();
mapper.enable(SerializationFeature.INDENT_OUTPUT);
String jsonString = mapper.writeValueAsString(result);
System.out.println(jsonString);
```

Future Enhancements!

- More client SDKs: C#, PHP

- Mapepire JDBC client

- TLS enhancements

- Logging and serviceability improvements

Takeaways

Config

Suitable performance

Always encrypted

Flexibility

Any Hardware

Any Language

Demo

Any Questions?

Important Links

Mapepire

Documentation   https://mapepire-ibmi.github.io/

Server Component   https://github.com/Mapepire-IBMi/mapepire-server

Node.js Client

GitHub Repository   https://github.com/Mapepire-IBMi/mapepire-js

NPM   https://www.npmjs.com/package/@ibm/mapepire-js

Java Client

GitHub Repository   https://github.com/Mapepire-IBMi/mapepire-java

Maven Central   https://central.sonatype.com/artifact/io.github.mapepire-ibmi/mapepire-sdk

Python Client

GitHub Repository   https://github.com/Mapepire-IBMi/mapepire-python

PyPi   https://pypi.org/project/mapepire-python/

Service Commander

GitHub Repository   https://github.com/ThePrez/ServiceCommander-IBMi

For More Information

IBM i
continuous innovation
continuous integration

Links You Need	Twitter	#Hashtags
IBM i Home Page: https://www.ibm.com/it-infrastructure/power/os/ibm-i (find link to Forrester Study and updated IBM i Strategy Whitepaper) IBM Strategy Whitepaper: https://www.ibm.com/it-infrastructure/us-en/resources/power/i-strategy-roadmap/ IBM Client Success: https://www.ibm.com/it-infrastructure/us-en/resources/power/ibm-i-customer-stories/ Support Life Cycle: https://www.ibm.com/support/lifecycle/ License Topics: https://www-01.ibm.com/support/docview.wss?uid=nas8N1022087 Fortra IBM i Marketplace Survey https://www.fortra.com/resources/guides/ibm-i-marketplace-survey-results	@IBMSystems @COMMONug @IBMChampions @IBMSystemsISVs @IBMiMag @ITJungleNews @SAPonIBMi @SiDforIBMi	#PowerSystems #IBMi #IBMAIX #POWER9 #LinuxonPower #OpenPOWER #HANAonPower #ITinfrastructure #OpenSource #HybridCloud #BigData

Forget ODBC! Here's a New Db2 Connector - Jonathan Zak and Sanjula Ganepola

Please take the last minute of this session to complete the evaluation. A direct link to the evaluation can be found using the QR code below.

COMMON
LEARN. ENGAGE. GROW.

56