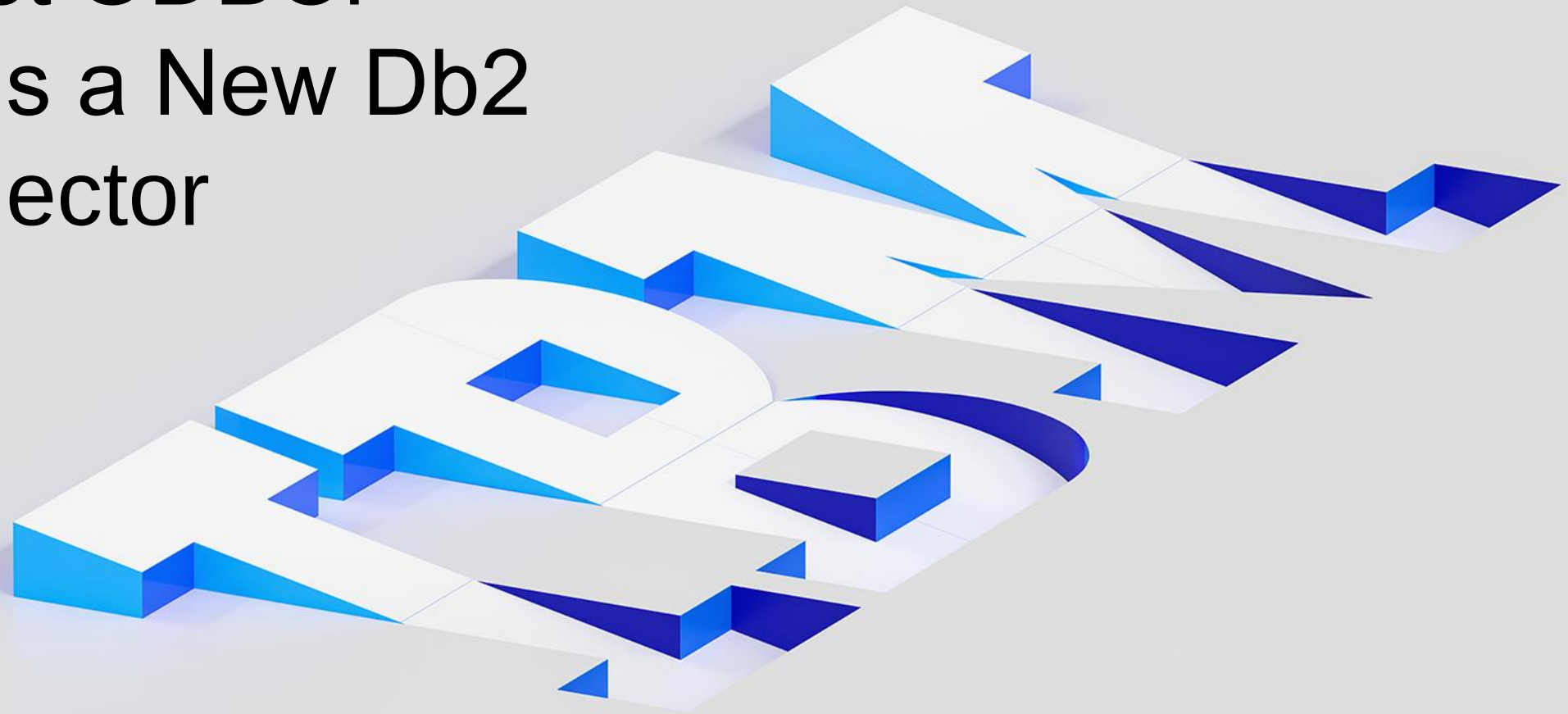
Forget ODBC! Here’s a New Db2 Connector



Jonathan Zak
Staff Software Engineer
Jonathan.Zak@ibm.com

Sanjula Ganepola
Software Developer
Sanjula.Ganepola@ibm.com

2025 February

IBM i

Agenda

- Mapepire Overview
- Architecture and Core Tenets
- Comparisons versus JDBC and ODBC
- Deep dive into the Java 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™

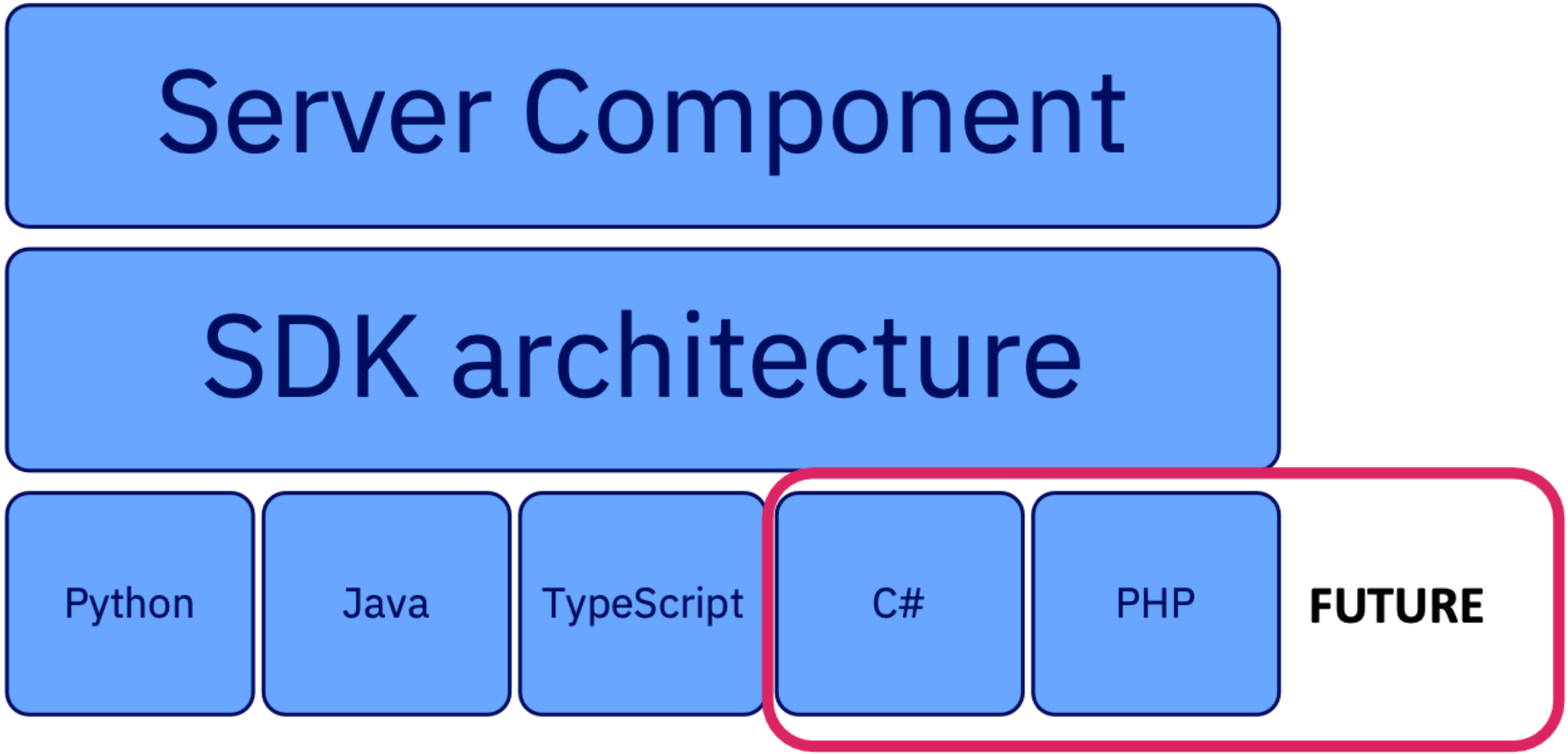
C#

php

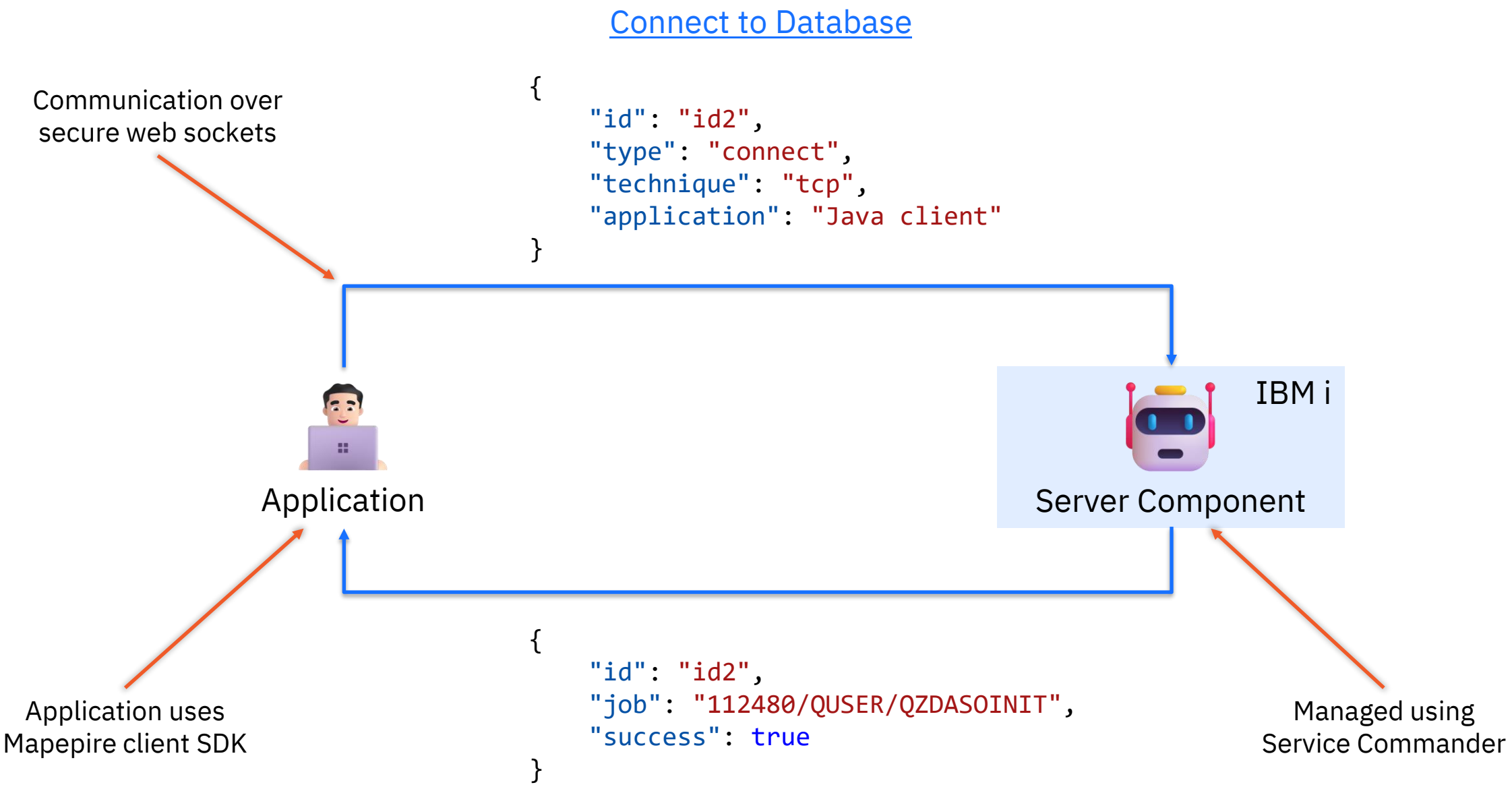
Mapepire Origin Story...

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



How does it work?



How does it work?

Query the Database

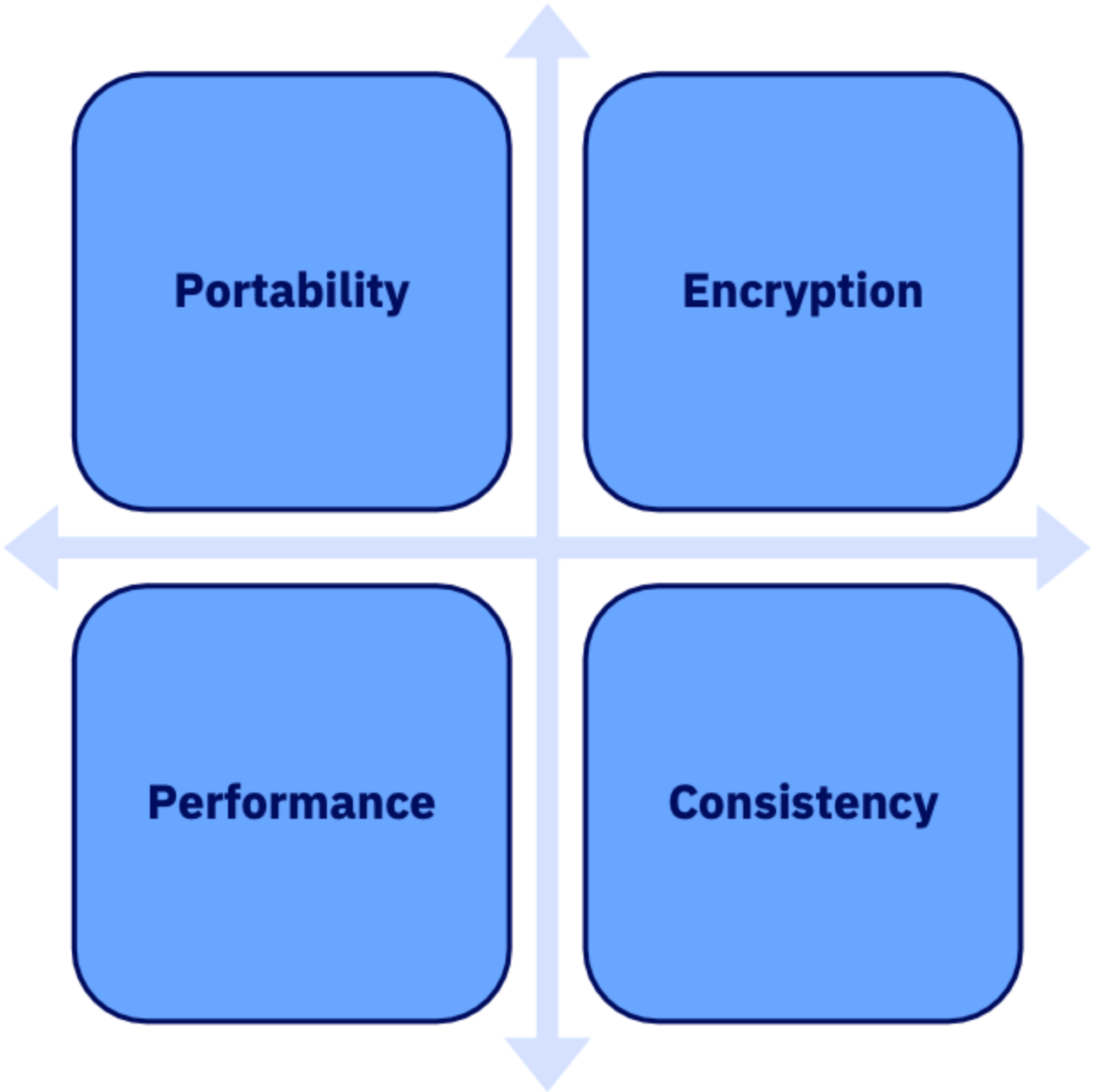
Request

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

Response

```
{
    "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
}
```

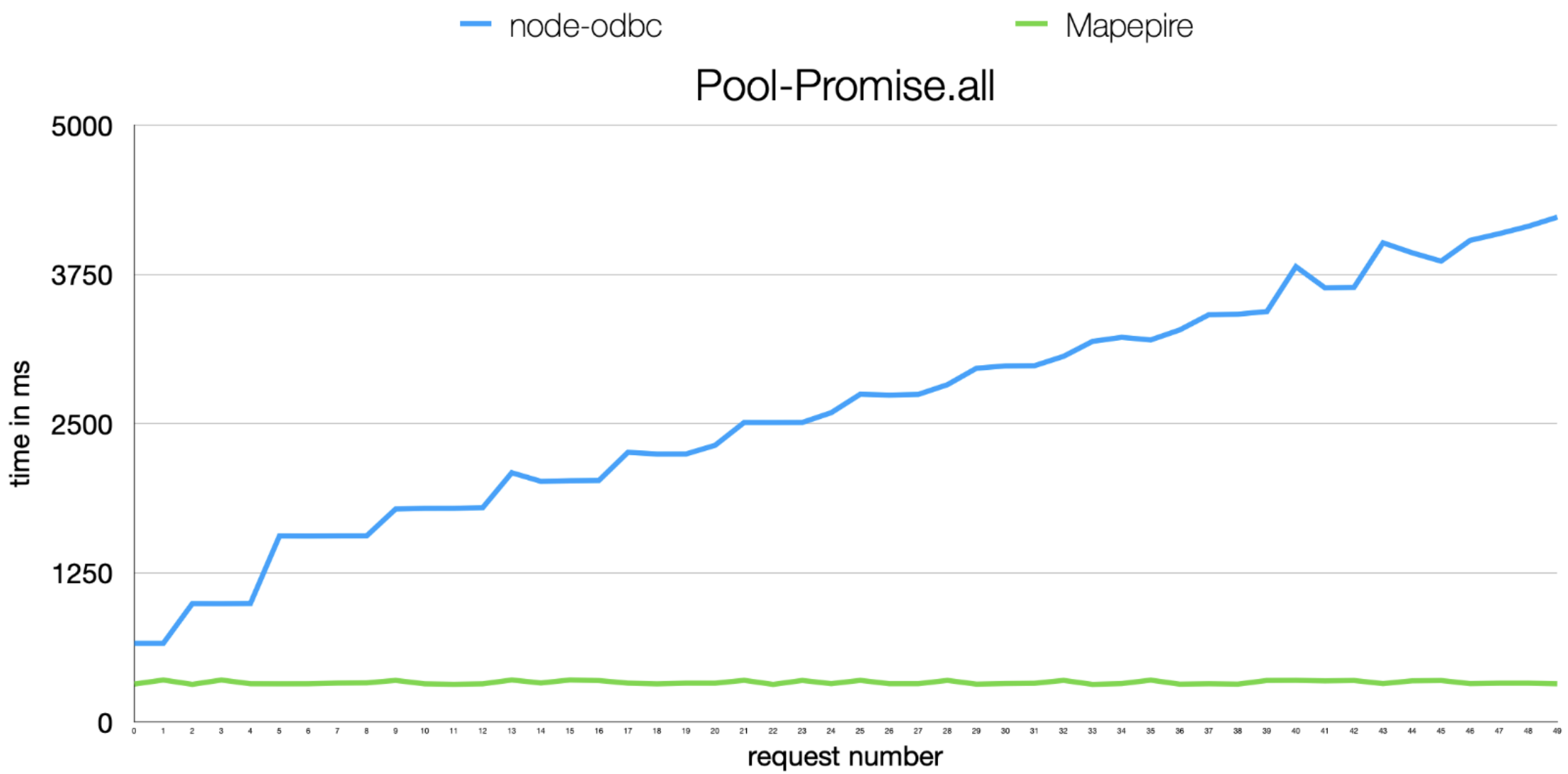
Core Tenets



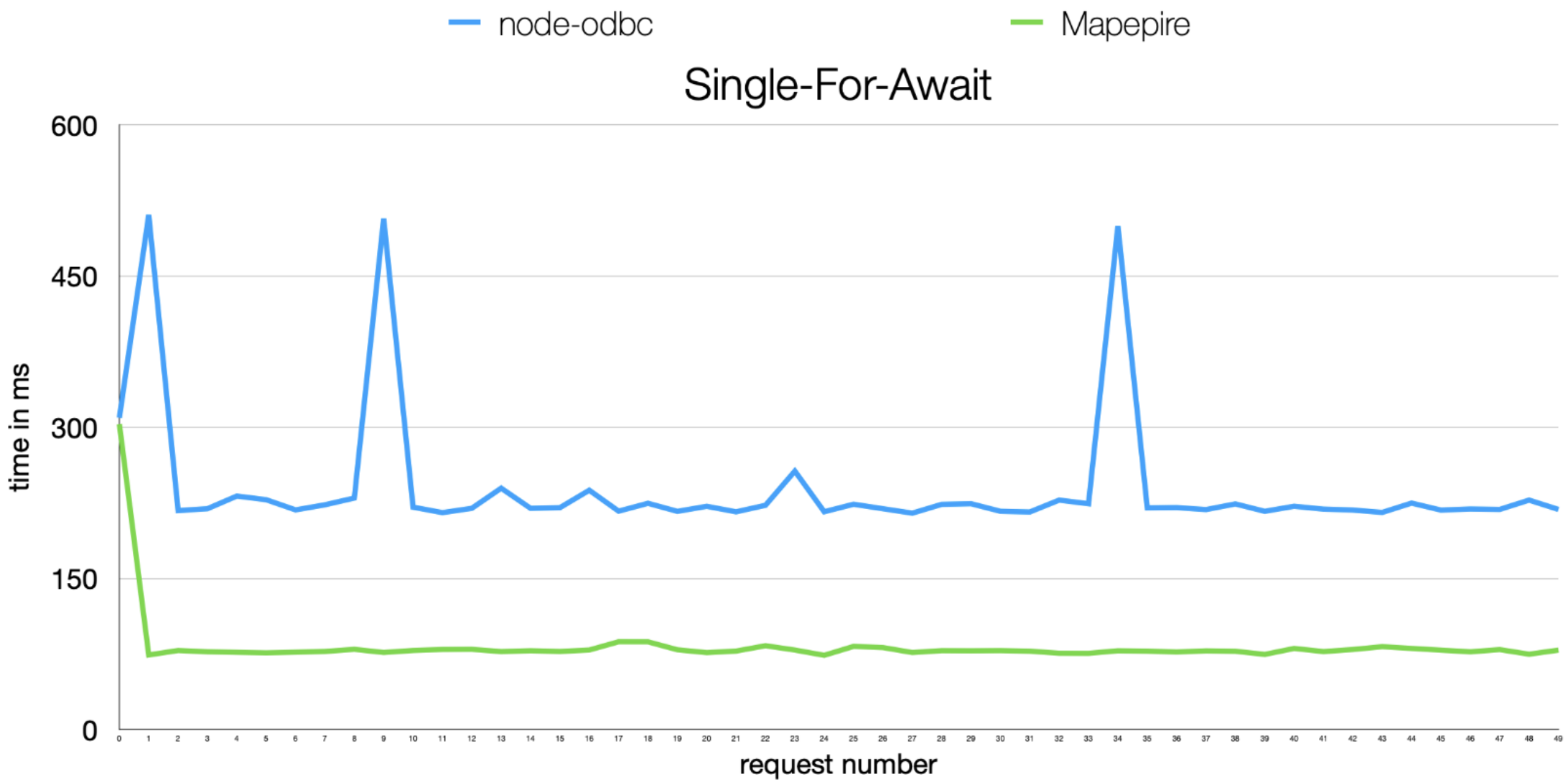
The biggest benefit of Mapepire.... Portability!!

	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



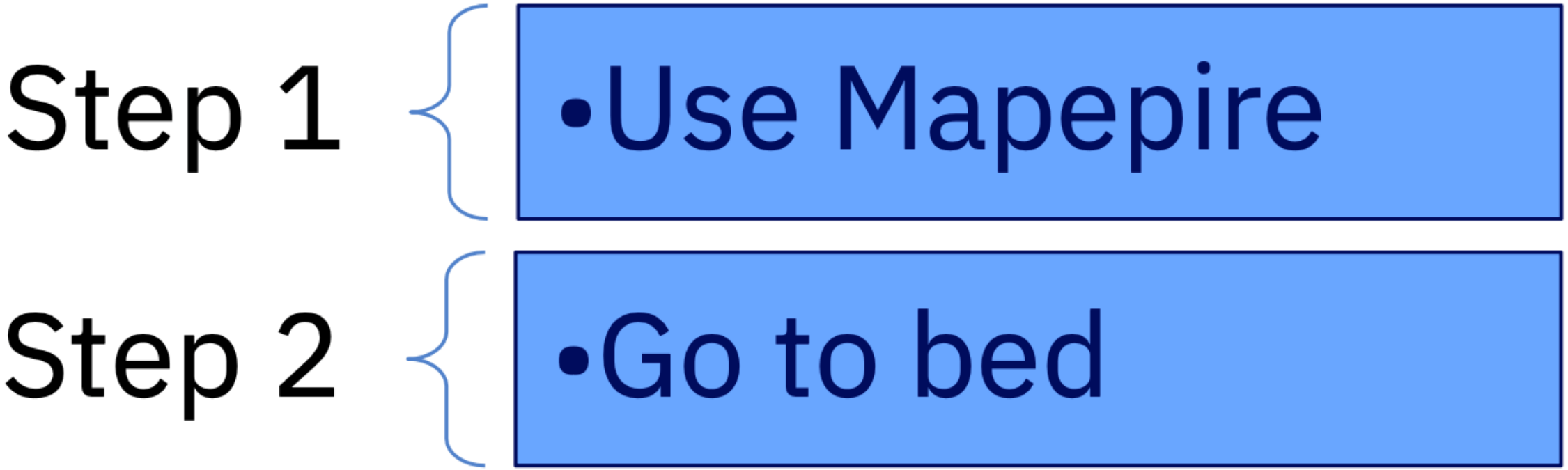
Some performance comparisons



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?



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

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

User profile and IP filtering

/QOpenSys/etc/mapepire/iprules.conf can be used govern which user profiles and IP addresses are able to connect

iprules.conf

QOpenSys > etc > mapepire > iprules.conf

```
1   # Allow connections from all hosts
2   allow *@*
3
4   # Disable connections for Q* user profiles
5   deny q*@*
```

```
// Create a single job and connect
DaemonServer creds = getDaemonServer();
SqlJob job = new SqlJob();
job.connect(creds).get();
```

⊗ java.util.concurrent.ExecutionException: java.sql.SQLException: java.io.IOException: Connection refused by security rule at line 5

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?

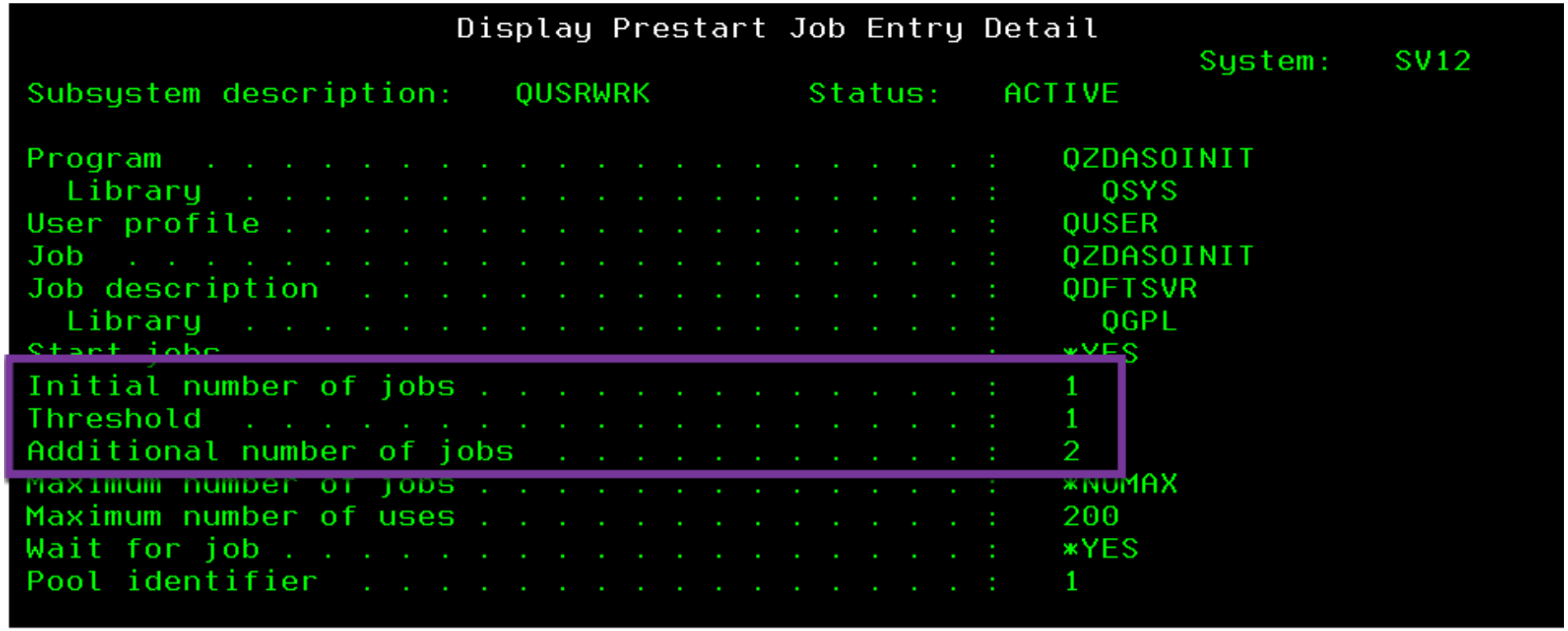
Separate jobs by application, user, etc.

- 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



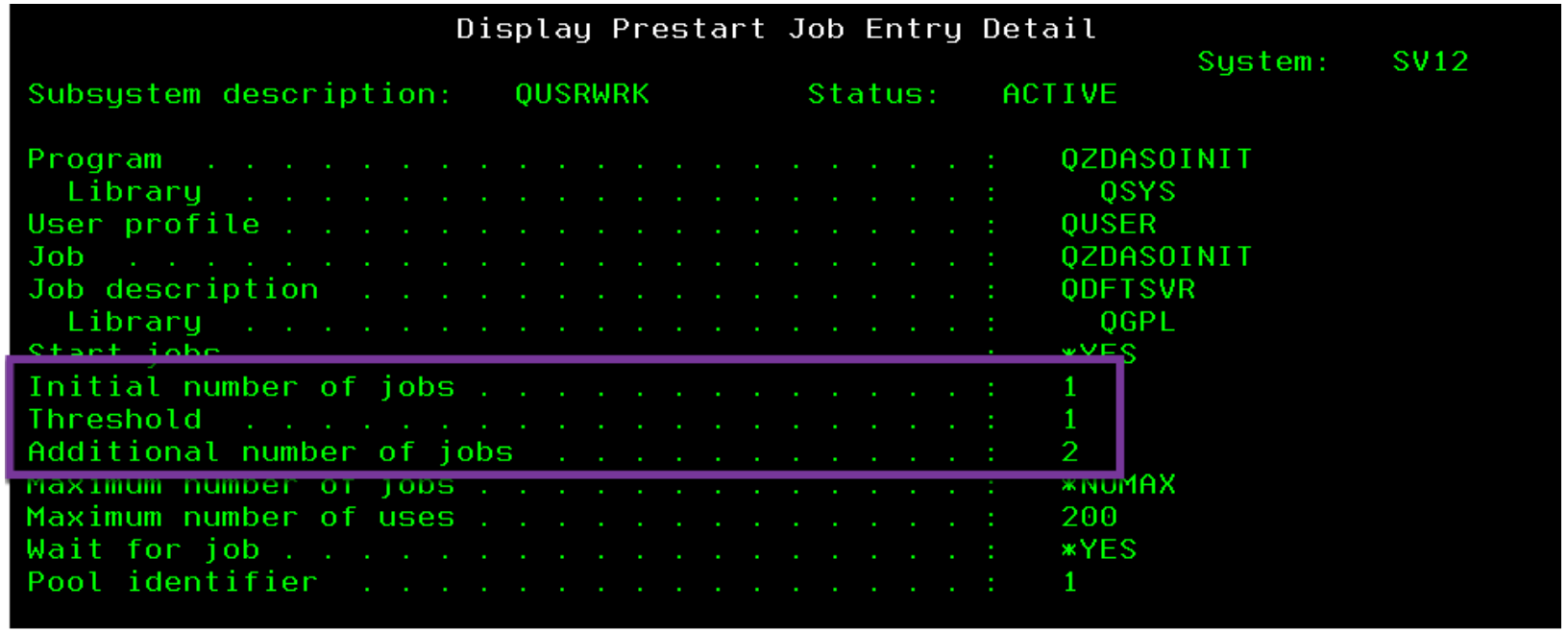
Default ODBC/JDBC prestart job settings

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)



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!

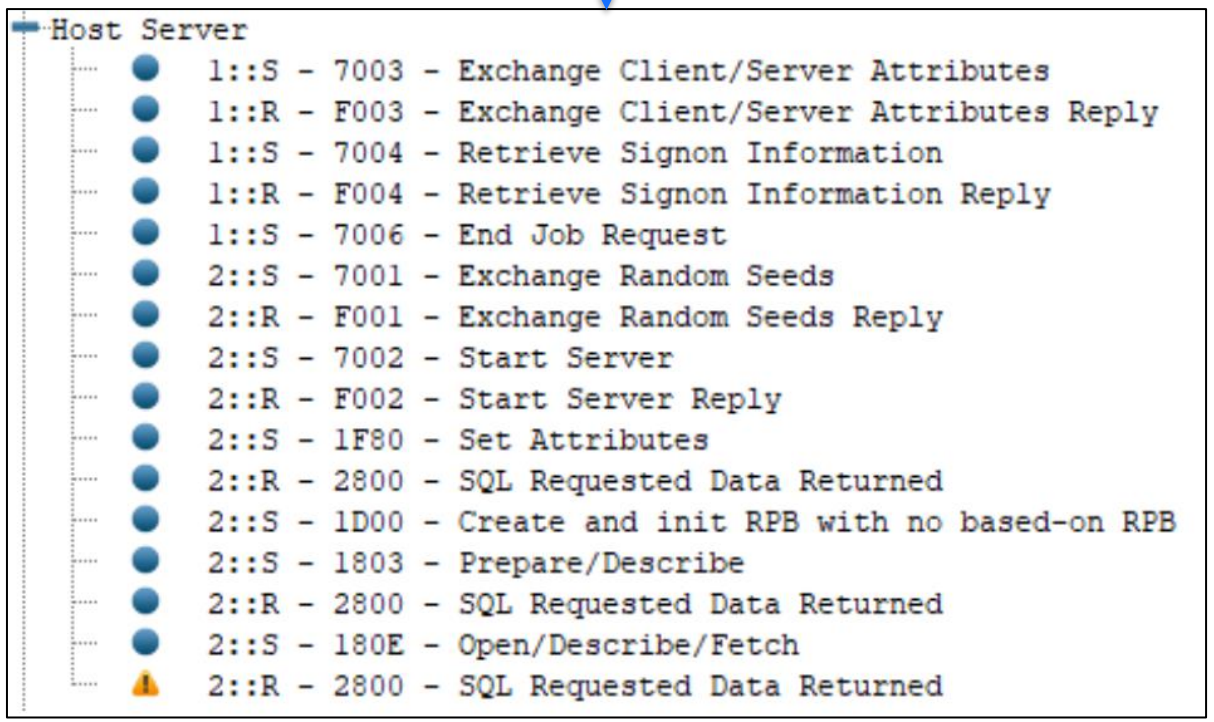
TCP distance to first database operation

JDBC/ODBC

Mapepire

Distinct TCP flows for a JDBC program!!

```java
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");
}
```



Meanwhile....

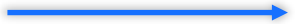
✨ Mapepire only needs 2 TCP flows

1. Connect and allocate a job
2. Run a query

Mapepire server setup

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

```
## Start
sc start mapepire

## Check it's running
sc check mapepire

## Stop
sc stop mapepire

## Check it's stopped
sc check mapepire
```



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

Mapepire-java client setup

Requirements

- Java 8 or later

Installation

- Add mapepire-sdk as a dependency (check for latest version on Maven Central Repository)

Links

- Usage Docs: https://mapepire-ibmi.github.io/guides/usage/java/
- GitHub Repository: https://github.com/Mapepire-IBMi/mapepire-java

README | Code of conduct | Apache-2.0 license

Mapepire Java Client SDK

Maven Central v0.1.1 | Maven Build passing | license Apache-2.0

Overview

`mapepire-java` is a Java client SDK that leverages the `mapepire-server` to provide a new and convenient way to access Db2 on IBM i.

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

```
<dependency>
    <groupId>io.github.mapepire-ibmi</groupId>
    <artifactId>mapepire-sdk</artifactId>
    <version>0.1.0</version>  <!-- Use the latest version -->
</dependency>
```

Setup credentials

Create a config.properties file with your IBM i connection

```
IBMI_HOST=host.somewhere.come
IBMI_USER=JIMBOB
IBMI_PASSWORD=letMeInNow
IBMI_PORT=8076
```

Create a helper function to construct a DaemonServer object

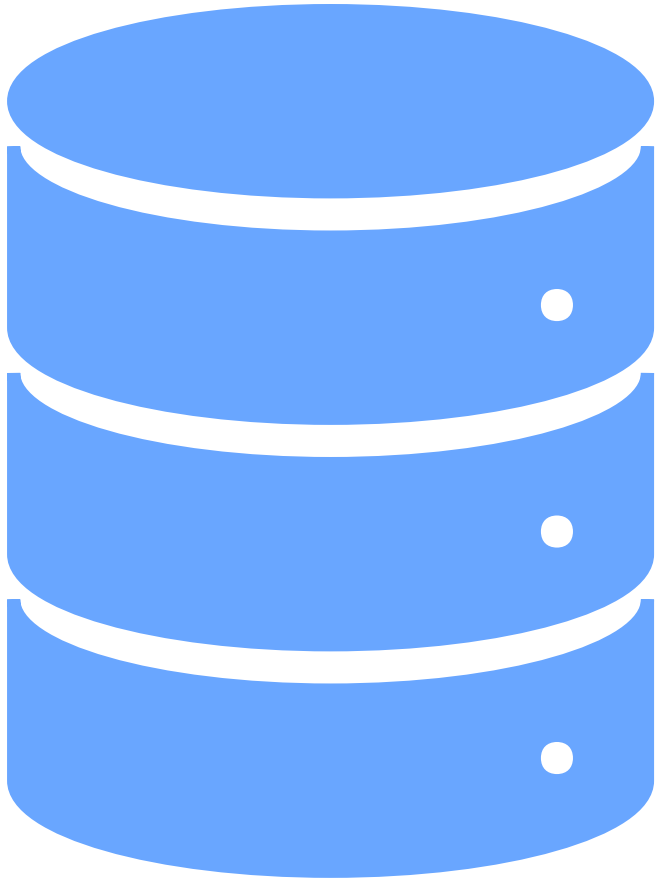
```
private static DaemonServer getDaemonServer() throws IOException {
    // Load config properties
    Properties properties = new Properties();
    try (InputStream input = App.class.getClassLoader().getResourceAsStream("config.properties")) {
        if (input == null) {
            throw new FileNotFoundException("Unable to find config.properties");
        }
        properties.load(input);
    }

    // Retrieve credentials
    String host = properties.getProperty("IBMI_HOST");
    String user = properties.getProperty("IBMI_USER");
    String password = properties.getProperty("IBMI_PASSWORD");
    int port = Integer.parseInt(properties.getProperty("IBMI_PORT"));

    return new DaemonServer(host, port, user, password, false, "");
}
```

Now it’s time to write some queries!

Here is some of the functionality:

- Connect to the database
- Run SQL statements
- Paginate results
- Run prepared statements
- Run a batch of statements
- Create and execute procedures
- Run CL commands
- Run SQL statements via a pool
- Leverage JDBC options
- Manage the server and JTOpen trace data
- Get Mapepire version
- Close a connection

... and much more!



Connecting to the database

```
// Create a single job and connect
DaemonServer creds = getDaemonServer();
SqlJob job = new SqlJob();
job.connect(creds).get();
```

Certificate configuration

Allow all certificates

```
DaemonServer creds = new DaemonServer("HOST", 8076, "USER", "PASSWORD", false);
```

Validate self-signed certificate

```
DaemonServer creds = new DaemonServer("HOST", 8076, "USER", "PASSWORD");
String ca = Tls.getCertificate(creds).get();
creds.setCa(ca);
```

Validate certificate signed by a recognized CA

```
DaemonServer creds = new DaemonServer("HOST", 8076, "USER", "PASSWORD");
```

Executing a query

```
// Create a single job and connect
DaemonServer creds = getDaemonServer();
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();

// Close query and job
query.close().get();
job.close();

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

Query result structure


```
{
  "id" : "query3",
  "success" : true,
  "error" : null,
  "sql_rc" : 0,
  "sql_state" : null,
  "execution_time" : 174,
  "metadata" : {
    "column_count" : 5,
    "columns" : [
      { "display_size" : 3, "label" : "DEPTNO", "name" : "DEPTNO", "type" : "CHAR", "precision" : 3, "scale" : 0 },
      { "display_size" : 36, "label" : "DEPTNAME", "name" : "DEPTNAME", "type" : "VARCHAR", "precision" : 36, "scale" : 0 },
      { "display_size" : 6, "label" : "MGRNO", "name" : "MGRNO", "type" : "CHAR", "precision" : 6, "scale" : 0 },
      { "display_size" : 3, "label" : "ADMRDEPT", "name" : "ADMRDEPT", "type" : "CHAR", "precision" : 3, "scale" : 0 },
      { "display_size" : 16, "label" : "LOCATION", "name" : "LOCATION", "type" : "CHAR", "precision" : 16, "scale" : 0 }
    ],
    "job" : "058971/QUSER/QZDASOINIT",
    "parameters" : null
  },
  "is_done" : false,
  "has_results" : true,
  "update_count" : -1,
  "data" : [
    { "DEPTNO" : "A00", "DEPTNAME" : "SPIFFY COMPUTER SERVICE DIV.", "MGRNO" : "000010", "ADMRDEPT" : "A00", "LOCATION" : null },
    { "DEPTNO" : "B01", "DEPTNAME" : "PLANNING", "MGRNO" : "000020", "ADMRDEPT" : "A00", "LOCATION" : null },
    { "DEPTNO" : "C01", "DEPTNAME" : "INFORMATION CENTER", "MGRNO" : "000030", "ADMRDEPT" : "A00", "LOCATION" : null }
  ],
  "parameter_count" : 0,
  "output_parms" : null
}
```

Paginating results

```
// Execute query and fetch 10 rows
Query query = job.query("SELECT * FROM SAMPLE.EMPLOYEE");
QueryResult<Object> result = query.execute(10).get();


// Continuously fetch 10 more rows until all all rows have been returned
while (!result.getIsDone()) {
    result = query.fetchMore(10).get();
}
```

Executing a prepared statement

```
// Prepare and execute query with parameters
QueryOptions options = new QueryOptions(false, false, Arrays.asList("A00"));
Query query = job.query("SELECT * FROM SAMPLE.DEPARTMENT WHERE ADMRDEPT = ?", options);
QueryResult<Object> result = query.execute().get();
```

Executing a batch of statements

```
QueryOptions options = new QueryOptions(false, false, Arrays.asList(
    Arrays.asList("SAM", "416 345 0879"),
    Arrays.asList("BOB", "647 821 7261"),
    Arrays.asList("JOHN", "289 726 1823"),
    Arrays.asList("JANE", "416 345 0879")
));
Query query = job.query("INSERT INTO SAMPLE.EMPLOYEE VALUES (?, ?)", options);
QueryResult<Object> result = query.execute().get();
```

Creating and executing a procedure

```
// Create a procedure
String testProc = String.join("\n", Arrays.asList(
        "CREATE OR REPLACE PROCEDURE SAMPLE.PROCEDURE_TEST("
                + "  IN P1 INTEGER,"
                + "  INOUT P2 INTEGER,"
                + "  OUT P3 INTEGER"
                + ")"
                + "BEGIN"
                + "  SET P3 = P1 + P2;"
                + "  SET P2 = 0;"
                + "END"));
Query queryA = job.query(testProc);
queryA.execute().get();

// Call the procedure with parameters
QueryOptions options = new QueryOptions(false, false, Arrays.asList(6, 4, 0));
Query queryB = job.query("CALL SAMPLE.PROCEDURE_TEST(?, ?, ?)", options);
QueryResult<Object> result = queryB.execute().get();
```

Leverage JDBC options

```
// Set JDBC options
JDBCOptions jdbcOptions = new JDBCOptions();
jdbcOptions.setNaming(Naming.SQL);
jdbcOptions.setLibraries(Arrays.asList("MYLIB1", "MYLIB2"));

// Create a single job with JDBC options
SqlJob job = new SqlJob(jdbcOptions);

// Create a pool with JDBC options
PoolOptions poolOptions = new PoolOptions(creds, jdbcOptions, 5, 3);
Pool pool = new Pool(poolOptions);
```

Executing a CL command

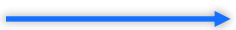
```
Query query = job.clCommand("CRTLIB LIB(MYLIB1) TEXT('My cool library')");
QueryResult<Object> result = query.execute().get();
```

Executing statements using a pool

```
// Create a pool with a max size of 5 and starting size of 3
DaemonServer creds = getDaemonServer();
PoolOptions poolOptions = new PoolOptions(creds, 5, 3);
Pool pool = new Pool(poolOptions);
pool.init().get();

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

// Close query and pool
query.close().get();
pool.end();
```

Retrieve the status of an SQL job

```
// Get the status of a job
SqlJob job = new SqlJob();
JobStatus status = job.getStatus();
```



```
/**
 * The job has not started yet.
 */
NotStarted(value:"notStarted"),

/**
 * The job is currently connecting to the server.
 */
Connecting(value:"connecting"),

/**
 * The job is ready to process queries.
 */
Ready(value:"ready"),

/**
 * The job is currently processing requests.
 */
Busy(value:"busy"),

/**
 * The job has ended.
 */
Ended(value:"ended");
```

Retrieve the state of a query

```
// Get the state of a query
Query query = job.query("SELECT * FROM SAMPLE.DEPARTMENT");
QueryState state = query.getState();
```



```
/**
 * Indicates that the query has not yet been
 run.
 */
NOT_YET_RUN(value:1),

/**
 * Indicates that the query has been
 executed, and more data is available for
 * retrieval.
 */
RUN_MORE_DATA_AVAILABLE(value:2),

/**
 * Indicates that the query has been
 successfully executed and all data has been
 * retrieved.
 */
RUN_DONE(value:3),

/**
 * Indicates that an error occurred during
 the query execution.
 */
ERROR(value:4);
```

Set trace configuration and retrieve trace data

```
// Send server trace data to a file and use the most verbose trace level
job.setTraceDest(ServerTraceDest.FILE).get();
job.setTraceLevel(ServerTraceLevel.DATASTREAM).get();

// Send JTOpen trace data to a file and use the most verbose trace level
job.setJtOpenTraceDest(ServerTraceDest.FILE).get();
job.setJtOpenTraceLevel(ServerTraceLevel.DATASTREAM).get();

// Get the trace file path and trace data
String traceFilePath = job.getTraceFilePath();
GetTraceDataResult result = job.getTraceData().get();
```

Server trace data

[INFO]: *2025-03-31.16.27.49.196*

```
Tracing enabled to file '/opt/mapepire/release/lib/mapepire/vsc-2025-03-31.16.27.49.191-99587952695951.html'
```

[ERR]: *2025-03-31.16.27.49.196*

```
com.ibm.as400.access.AS400JDBCSQLSyntaxErrorException: [SQL0601] MAPEPIRE_TEST in *LIBL type *LIB already exists.
        at com.ibm.as400.access.JDError.createSQLExceptionSubClass(JDError.java:941)
        at com.ibm.as400.access.JDError.throwSQLException(JDError.java:738)
        at com.ibm.as400.access.AS400JDBCStatement.commonPrepare(AS400JDBCStatement.java:1634)
        at com.ibm.as400.access.AS400JDBCStatement.execute(AS400JDBCStatement.java:2148)
        at com.github.ibm.mapepire.requests.RunSql.go(RunSql.java:22)
        at com.github.ibm.mapepire.ClientRequest.run(ClientRequest.java:87)
        at java.lang.Thread.run(Thread.java:825)
```

[ERR]: *2025-03-31.16.28.12.240*

```
com.ibm.as400.access.AS400JDBCSQLSyntaxErrorException: [SQL5016] Qualified object name DEPARTMENT not valid.
        at com.ibm.as400.access.JDError.createSQLExceptionSubClass(JDError.java:941)
        at com.ibm.as400.access.JDError.throwSQLException(JDError.java:738)
        at com.ibm.as400.access.AS400JDBCStatement.commonPrepare(AS400JDBCStatement.java:1840)
        at com.ibm.as400.access.AS400JDBCStatement.execute(AS400JDBCStatement.java:2148)
        at com.github.ibm.mapepire.requests.RunSql.go(RunSql.java:22)
        at com.github.ibm.mapepire.ClientRequest.run(ClientRequest.java:87)
        at java.lang.Thread.run(Thread.java:825)
```

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

```
// 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

```
// 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);
```

Demo

Takeaways

Config	Great performance
	Always encrypted
Flexibility	Any Hardware
	Any Language

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

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

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 to the right.

IBM