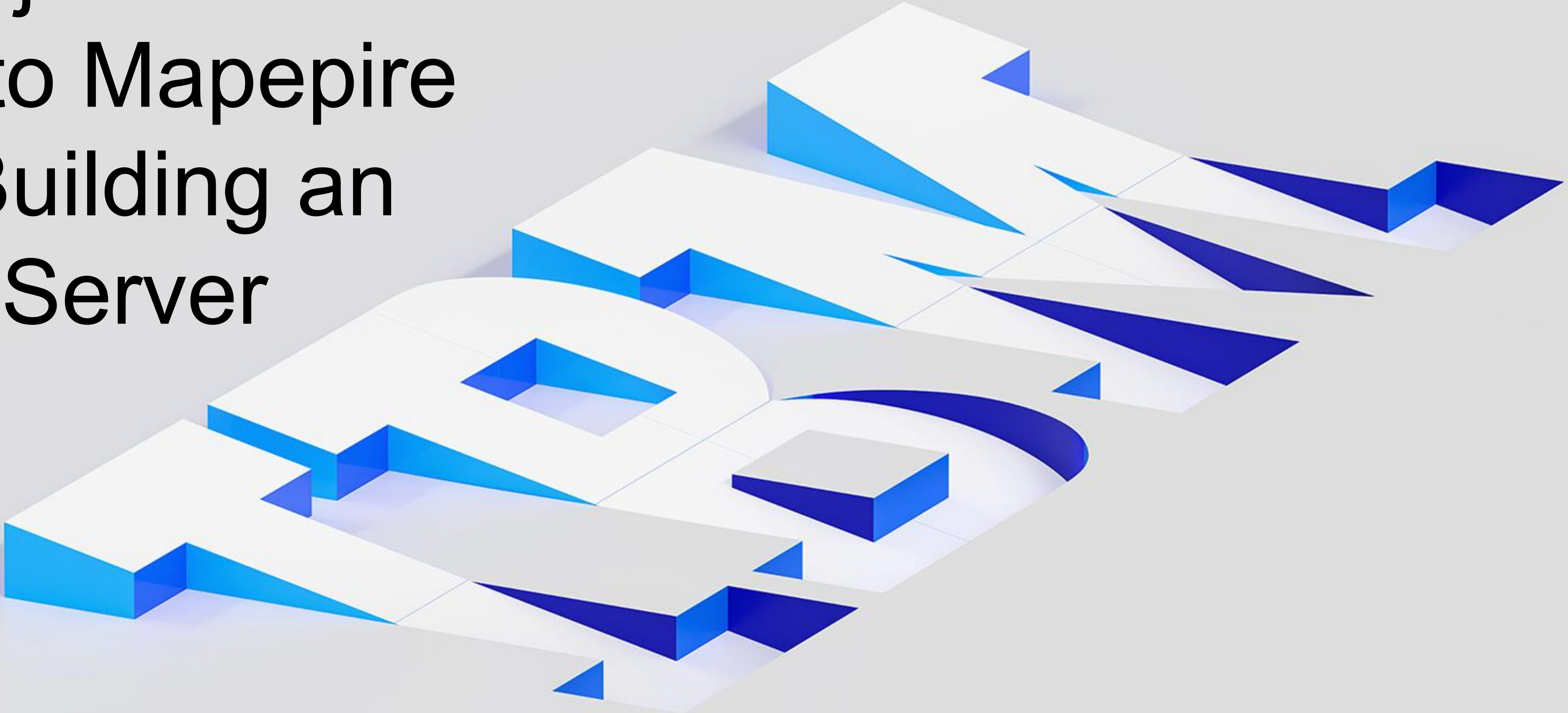
Node.js In Action - Intro to Mapepire and Building an MCP Server



Sanjula Ganepola
Software Developer – IBM i App Dev & AI Toolchain
Sanjula.Ganepola@ibm.com

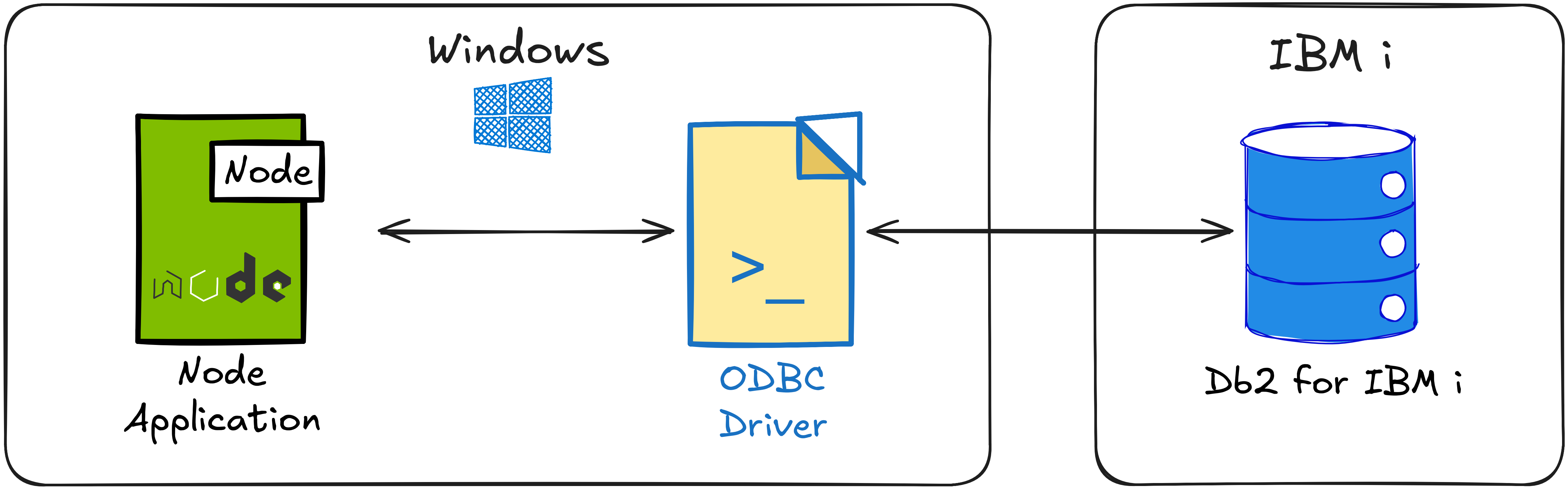
2025 September

Agenda

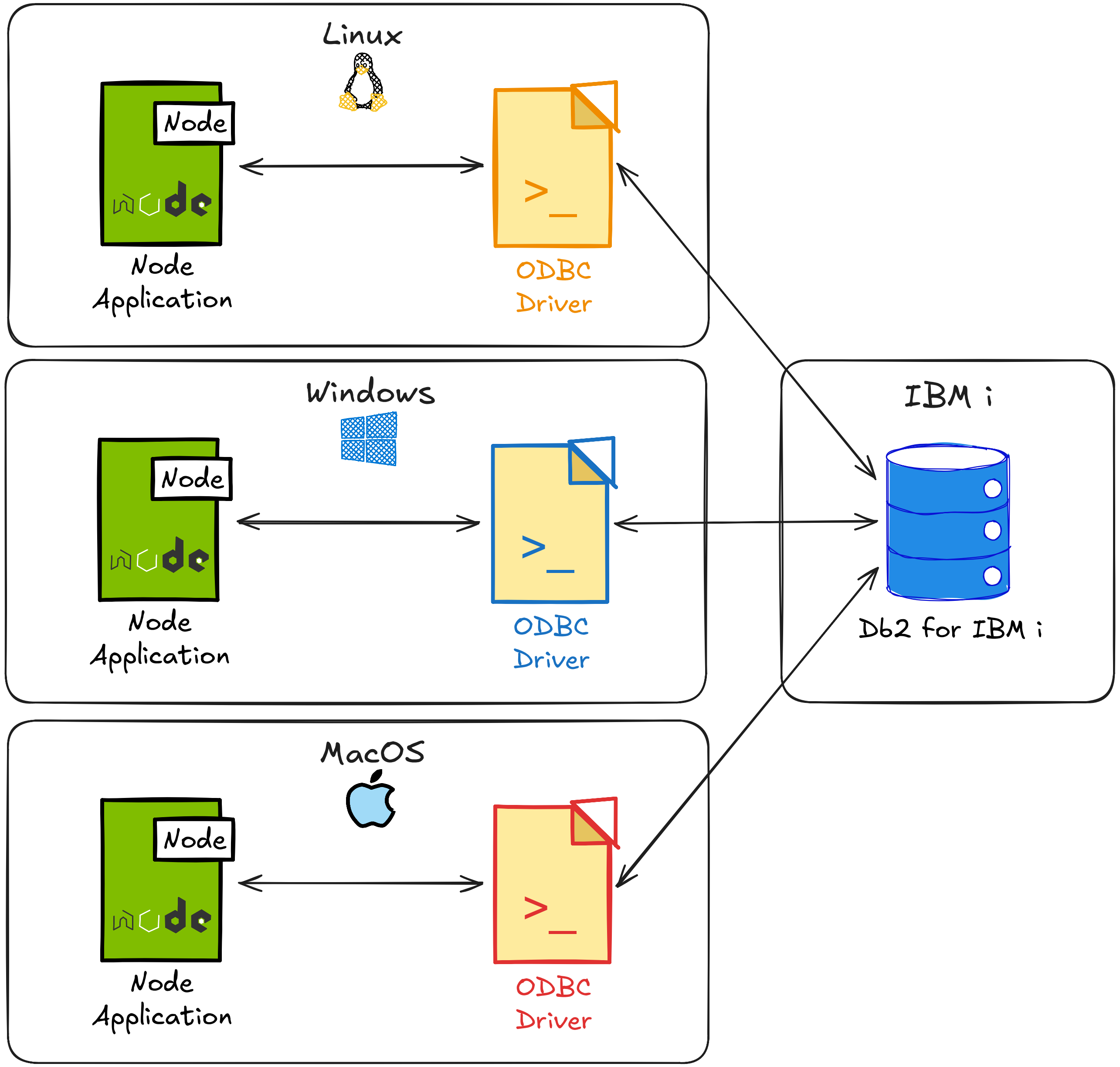
- Problems with node-odbc
- Building Node.js Applications with Mapepire
  - Mapepire Overview
  - Benefits and architecture
  - ConnectMe REST server demo
- Building an MCP Server
  - Important Concepts
  - ConnectMe MCP server demo

Problems with node-odbc

Let’s start simple with a node-odbc on Windows

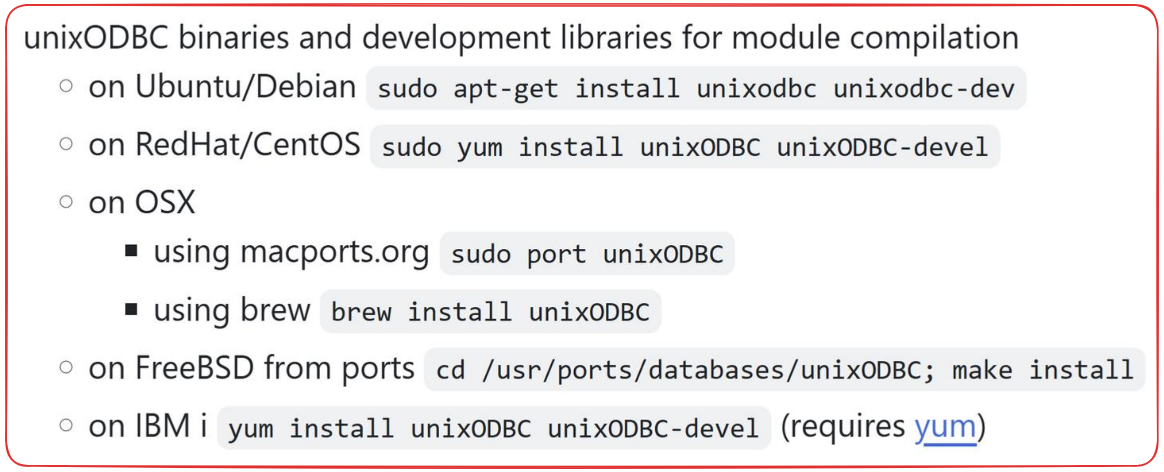


But now I need platform specific drivers



node-odbc is also not fun to setup

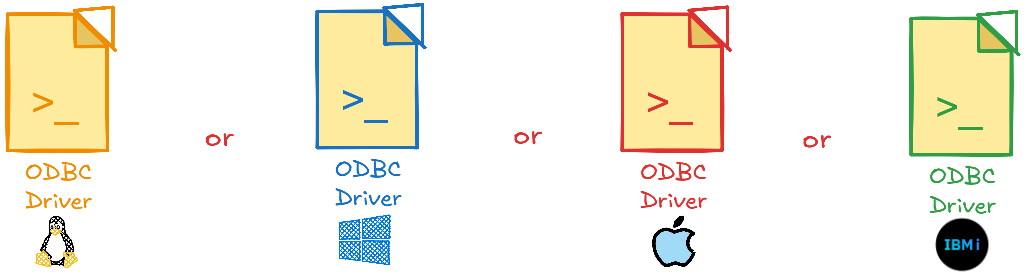
1. Install unixODBC →



unixODBC binaries and development libraries for module compilation

- on Ubuntu/Debian `sudo apt-get install unixodbc unixodbc-dev`
- on RedHat/CentOS `sudo yum install unixODBC unixODBC-devel`
- on OSX
  - using macports.org `sudo port unixODBC`
  - using brew `brew install unixODBC`
- on FreeBSD from ports `cd /usr/ports/databases/unixODBC; make install`
- on IBM i `yum install unixODBC unixODBC-devel` (requires yum)

2. Install ODBC driver →



3. Edit configuration file →

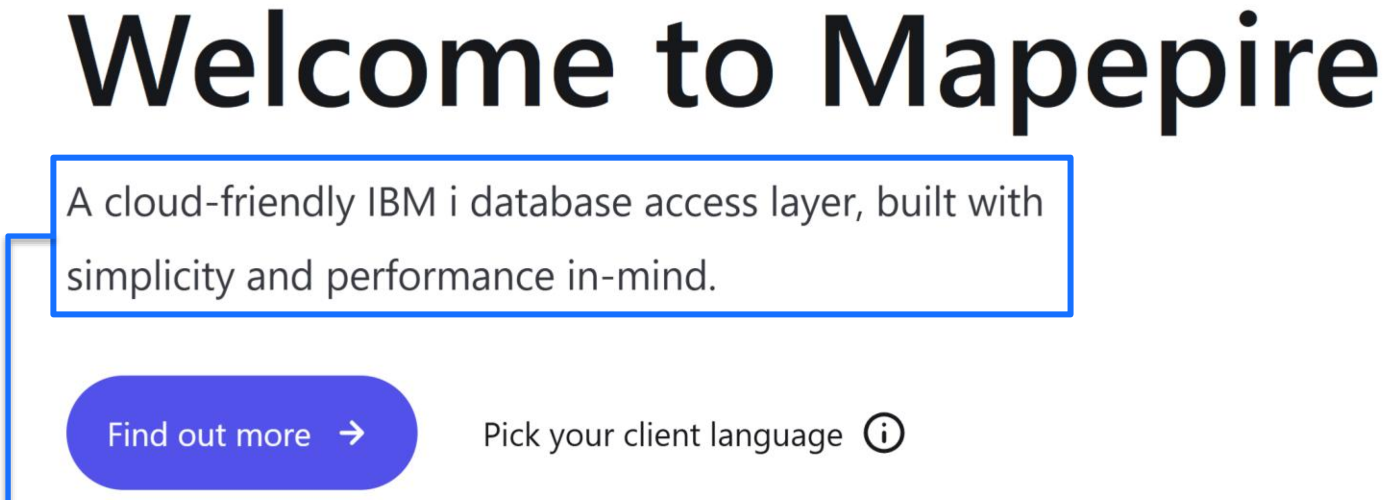
```
[Db2]
Description = Db2 Driver
Driver = ...
fileusage=1
dontdlclose=1
```

4. Install dependencies in Node Application →

```
npm i odbc
```

Building Node.js Applications with Mapepire

What is Mapepire?



Welcome to Mapepire

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

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

Mapepire Origin Story...

January 2020

- VSCode "Code for IBM i" extension includes basic Db2 support

February 2022

- Work begins on Server component to power Db2 features in VSCode

March 2022

- First release of VSCode Db2 for i extension

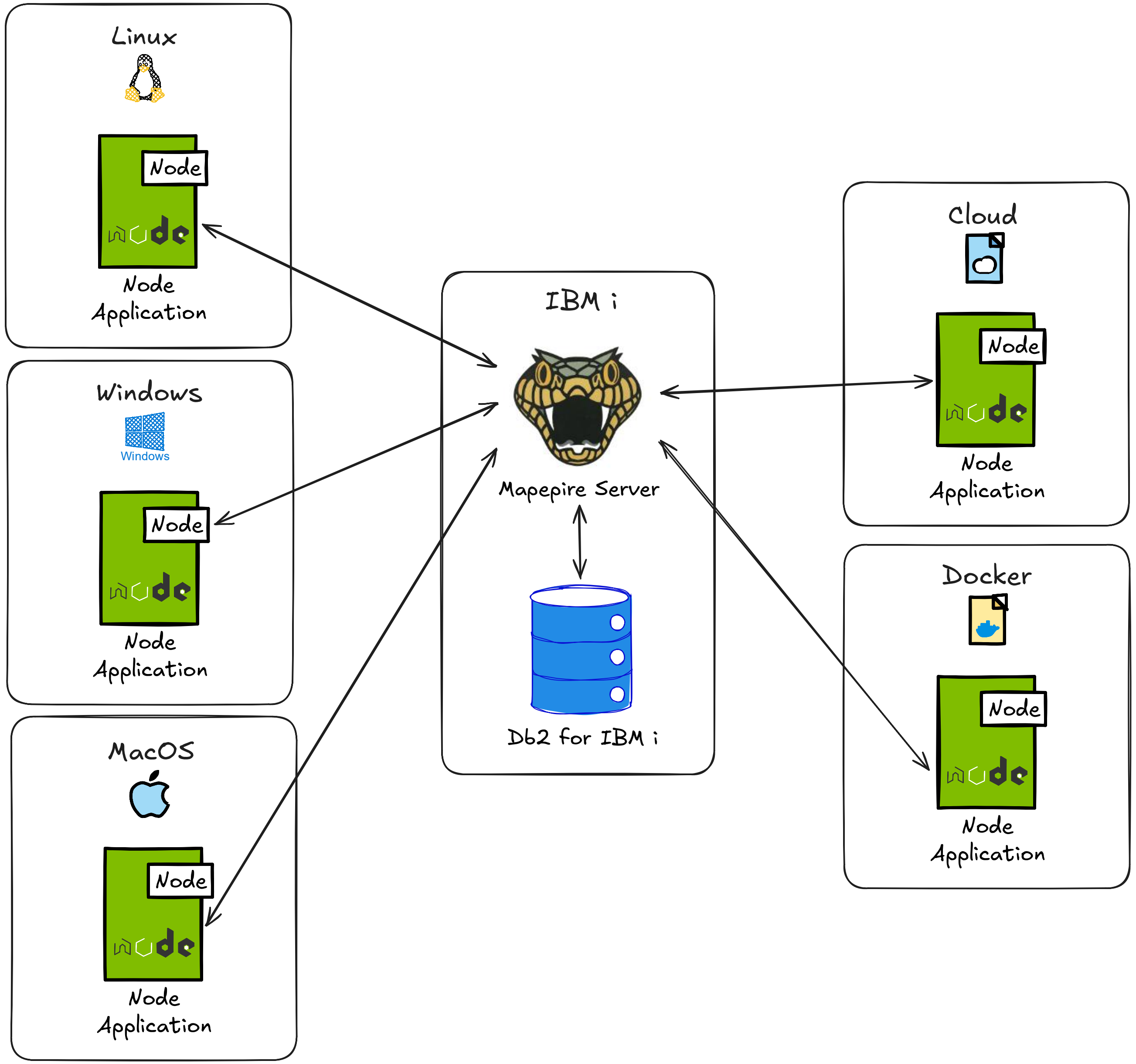
July 2023

- VSCode Db2 for i extension publishes v0.3.0, the first release leveraging server component (v0.3.0)

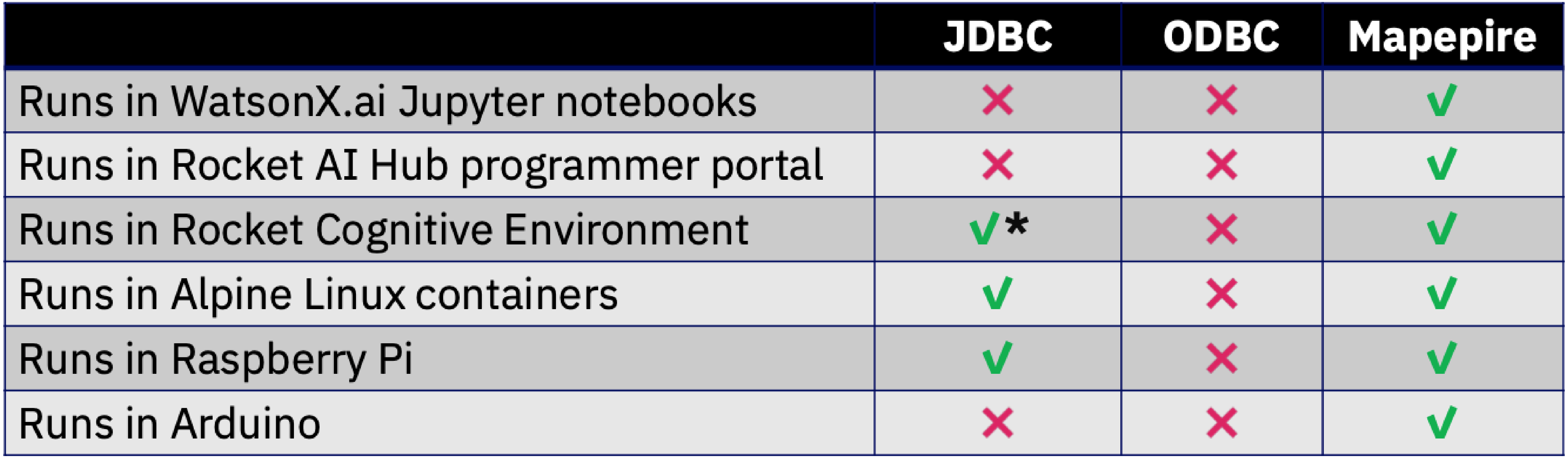
August 2024

- Mapepire is born!

Cloud-friendly

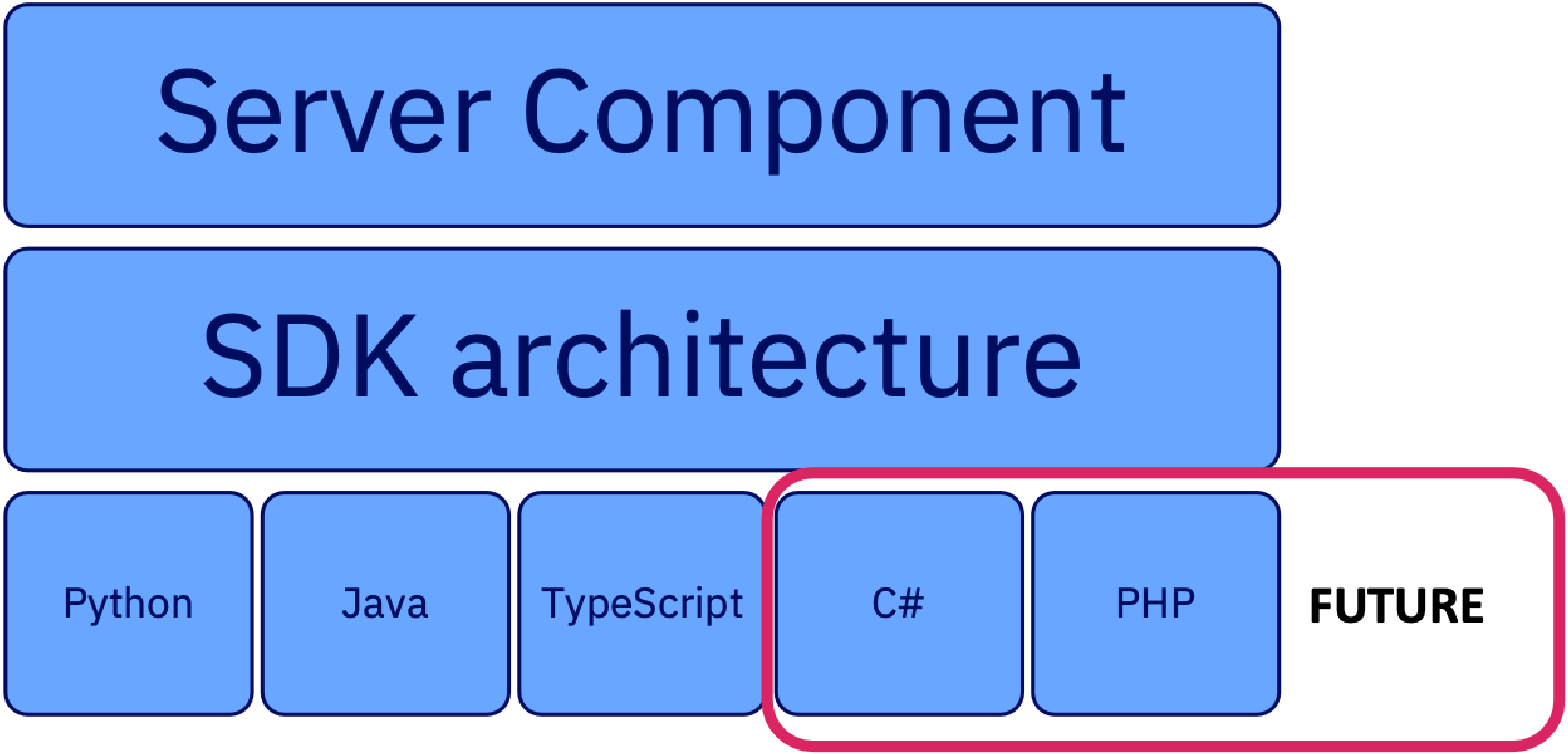


Run your apps anywhere!



	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	✗	✗	✓

Simplicity



Server Component

SDK architecture

Python

Java

TypeScript

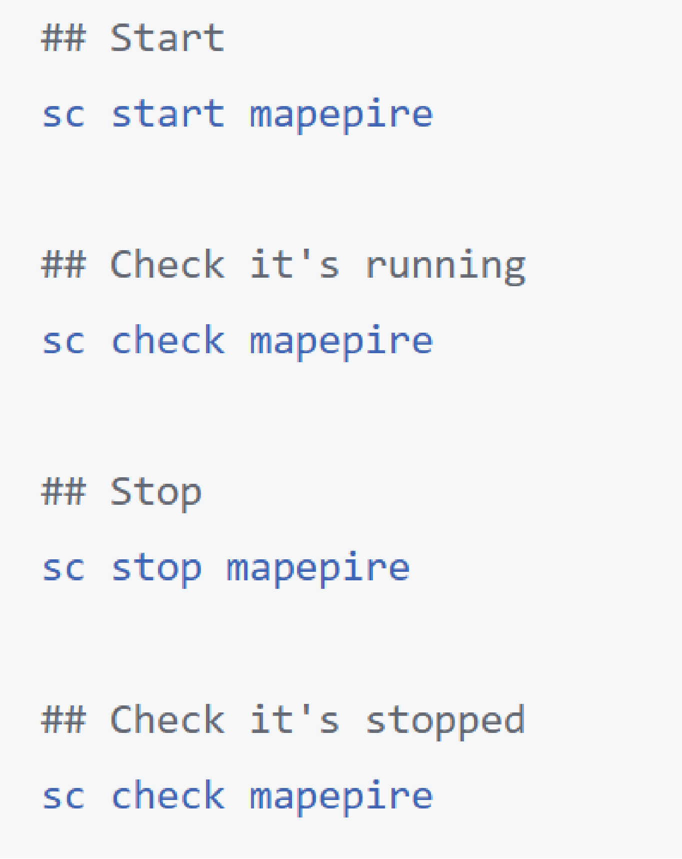
C#

PHP

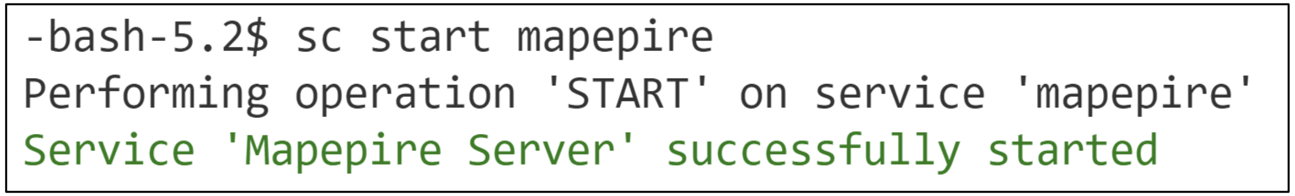
FUTURE

Mapepire server setup

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



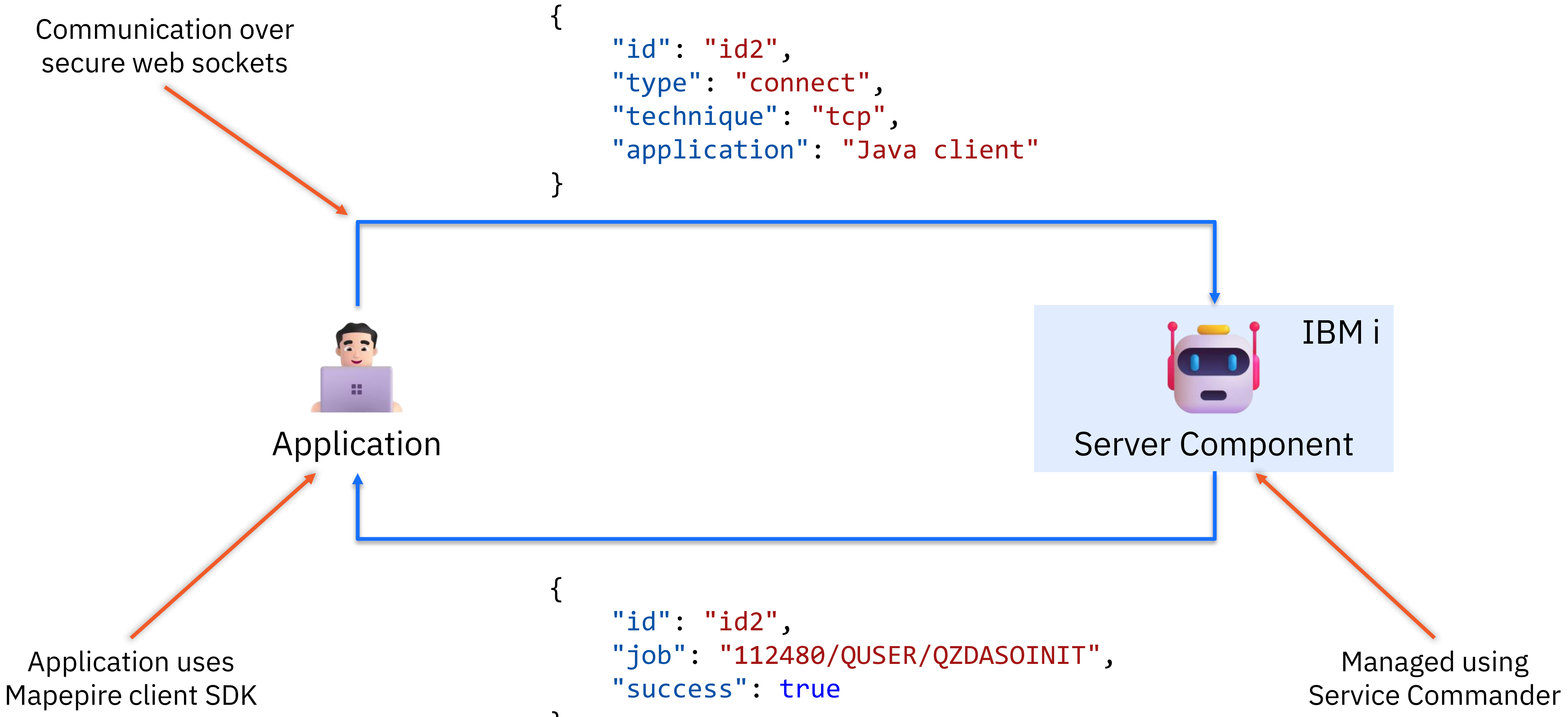
```
## Start
sc start mapepire

## 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
```

How does an application talk to the Mapepire Server?

Connect to Database



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

Communication over secure web sockets

Application

IBM i

Server Component

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

Application uses Mapepire client SDK

Managed using Service Commander

Another example of a JSON exchange

Query the Database

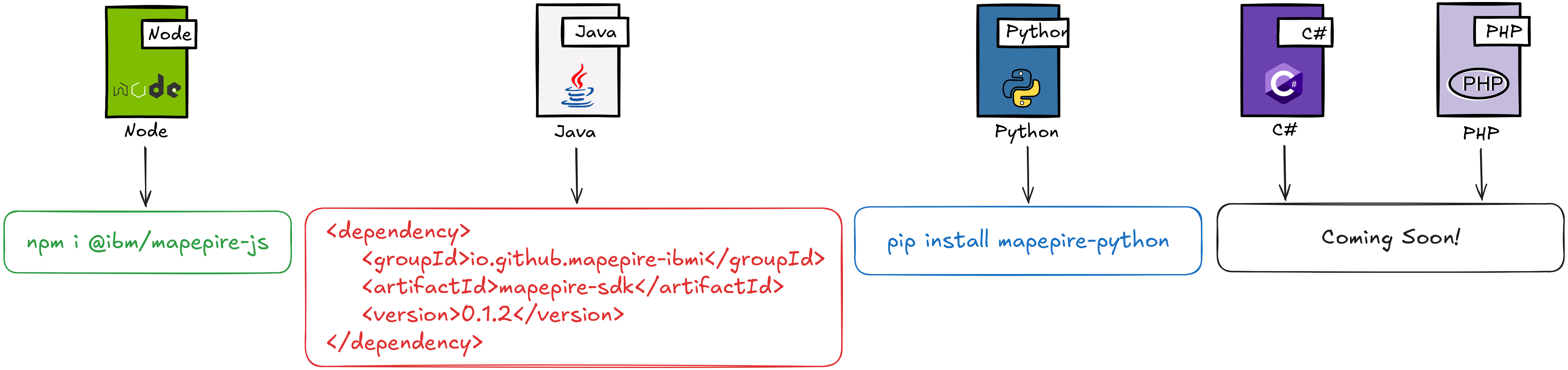
Request

```
{
    "id": "query3",
    "type": "sql",
    "sql": "SELECT * FROM SAMPLE.DEPARTMENT WHERE
DEPTNAME = 'SPIFFY COMPUTER SERVICE DIV.'",
    "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
}
```

The Mapepire SDKs make it easy



Node

```
npm i @ibm/mapepire-js
```

Java

```
<dependency>
    <groupId>io.github.mapepire-ibmi</groupId>
    <artifactId>mapepire-sdk</artifactId>
    <version>0.1.2</version>
</dependency>
```

Python

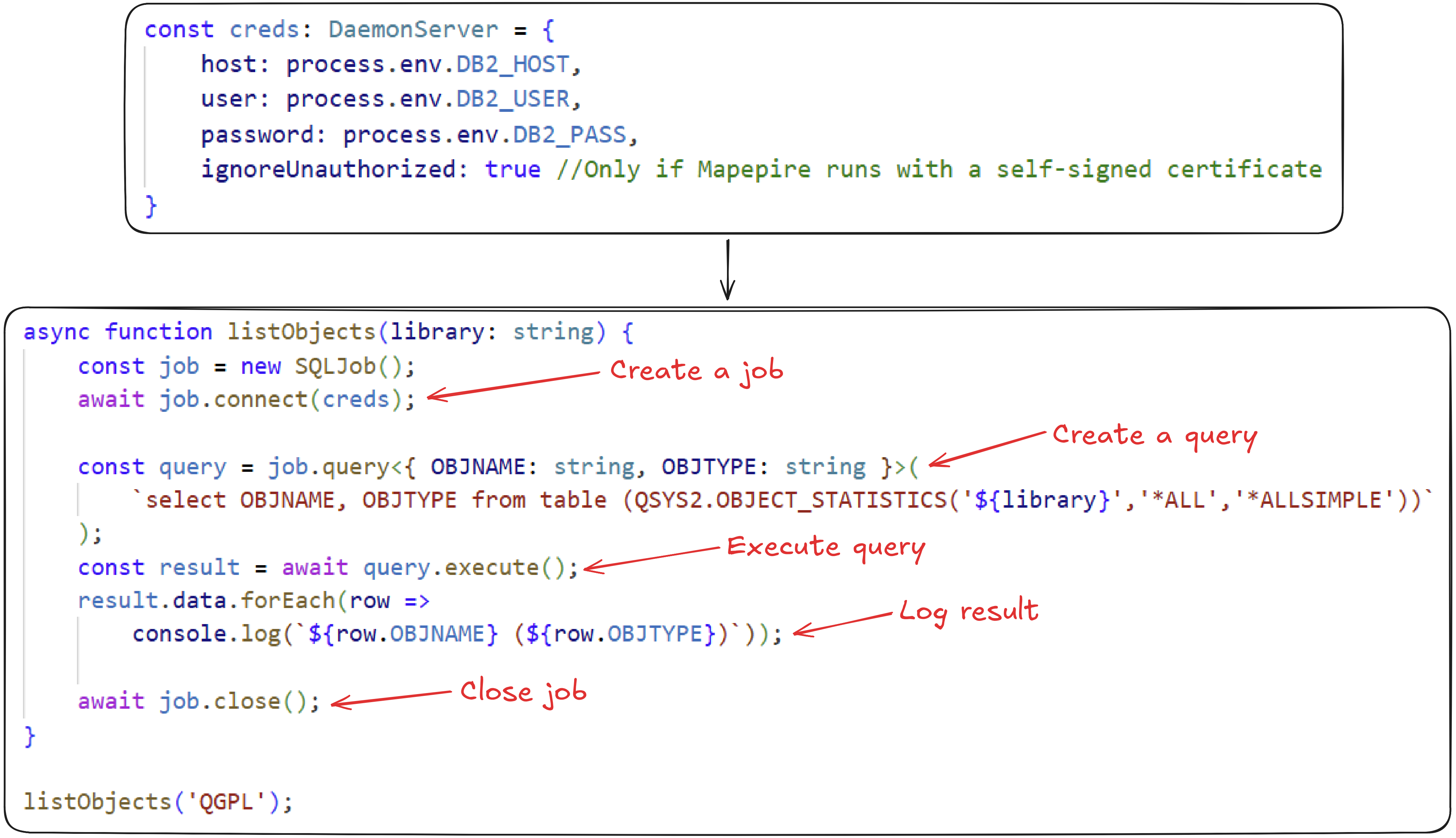
```
pip install mapepire-python
```

C#

PHP

Coming Soon!

Making a connection and running a query



```
const creds: DaemonServer = {
    host: process.env.DB2_HOST,
    user: process.env.DB2_USER,
    password: process.env.DB2_PASS,
    ignoreUnauthorized: true //Only if Mapepire runs with a self-signed certificate
}
```

```
async function listObjects(library: string) {
    const job = new SQLJob();
    await job.connect(creds);

    const query = job.query<{ OBJNAME: string, OBJTYPE: string }>(
        `select OBJNAME, OBJTYPE from table (QSYS2.OBJECT_STATISTICS('${library}','*ALL','*ALLSIMPLE'))`
    );
    const result = await query.execute();
    result.data.forEach(row =>
        console.log(`${row.OBJNAME} (${row.OBJTYPE})`));

    await job.close();
}

listObjects('QGPL');
```

Create a job

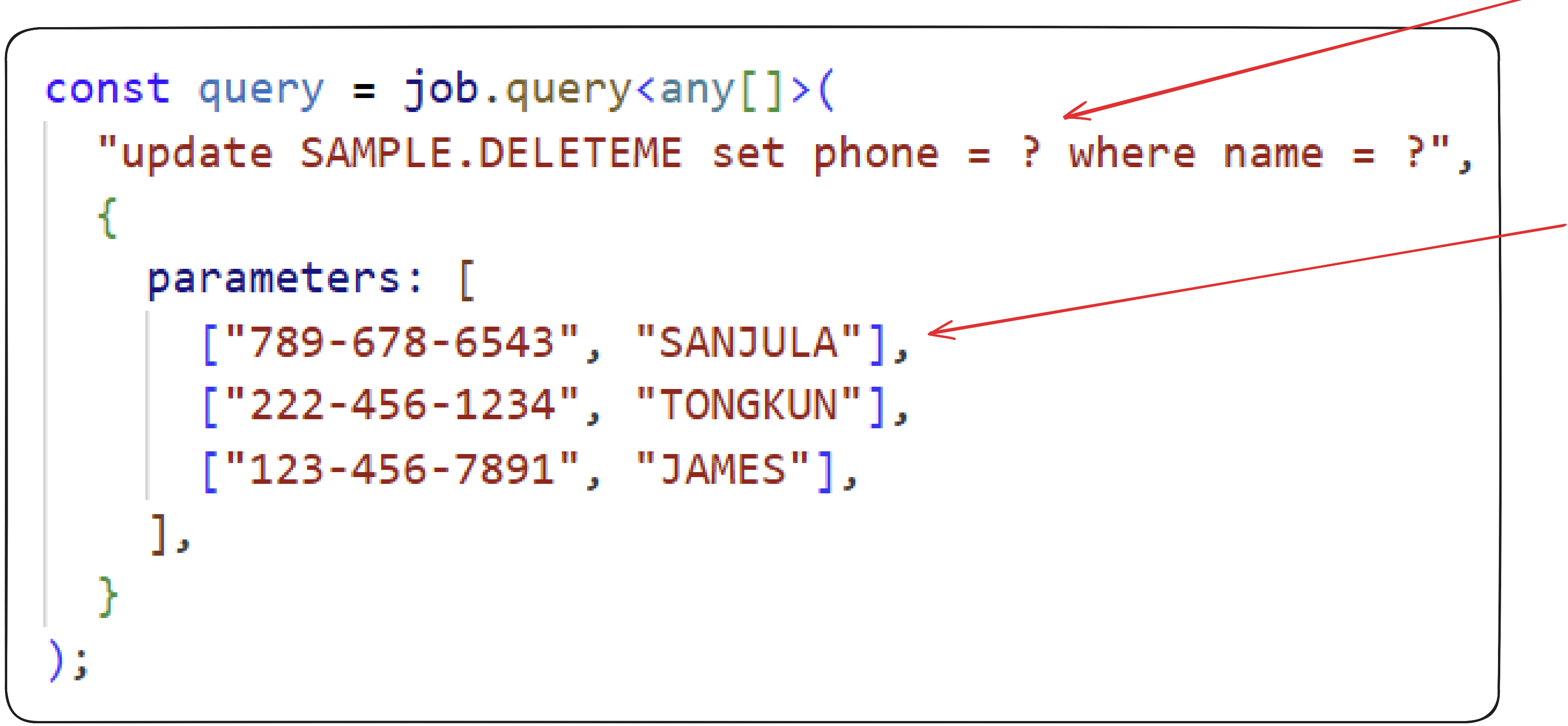
Create a query

Execute query

Log result

Close job

Using prepared statements and batch parameters



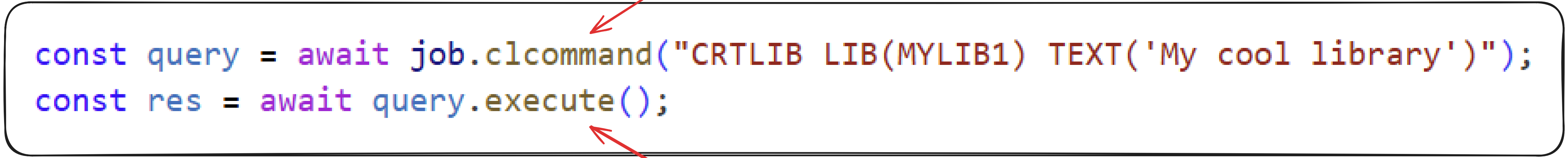
```
const query = job.query<any[]>(
  "update SAMPLE.DELETEME set phone = ? where name = ?",
  {
    parameters: [
      ["789-678-6543", "SANJULA"],
      ["222-456-1234", "TONGKUN"],
      ["123-456-7891", "JAMES"],
    ],
  }
);
```

Create prepared statement

Pass parameters in batch

Running CL commands

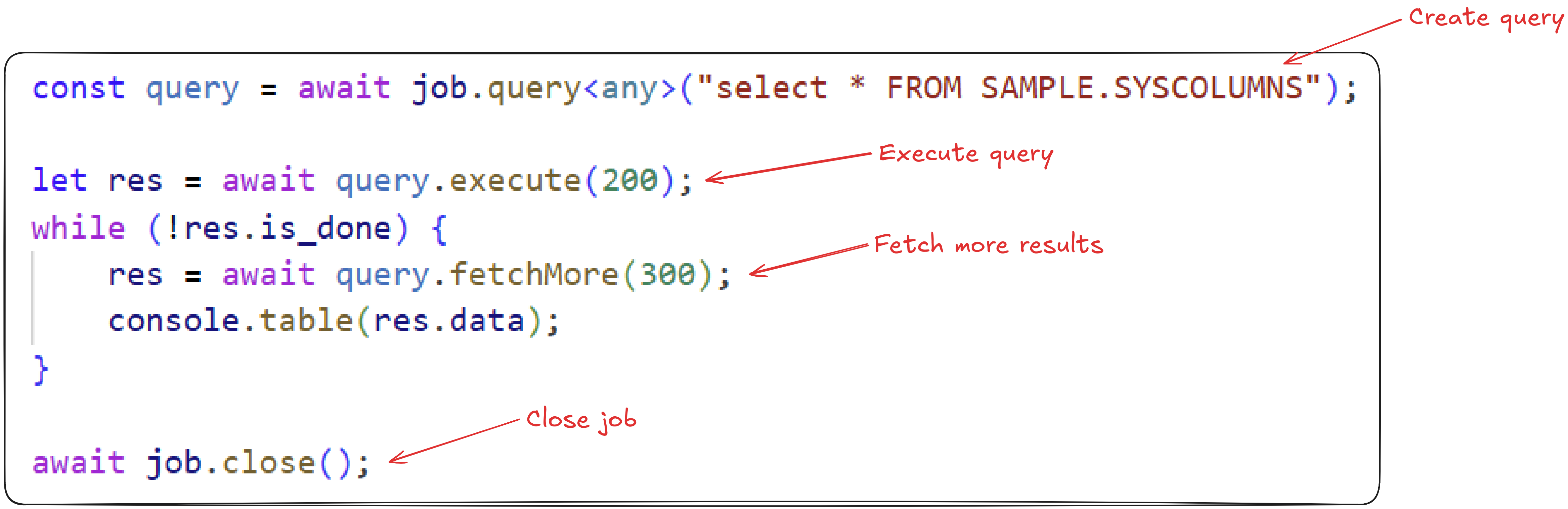
Create query
with CL command



```
const query = await job.clcommand("CRTLIB LIB(MYLIB1) TEXT('My cool library')");
const res = await query.execute();
```

Execute
CL command

Paging results



```
const query = await job.query<any>("select * FROM SAMPLE.SYSCOLUMNS");

let res = await query.execute(200);
while (!res.is_done) {
    res = await query.fetchMore(300);
    console.table(res.data);
}

await job.close();
```

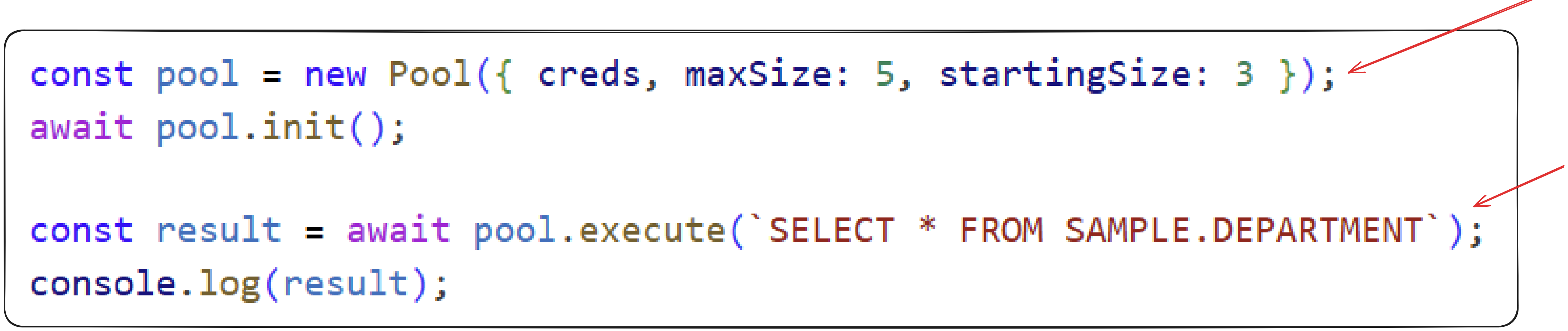
Create query

Execute query

Fetch more results

Close job

Job pooling



```
const pool = new Pool({ creds, maxSize: 5, startingSize: 3 });
await pool.init();

const result = await pool.execute(`SELECT * FROM SAMPLE.DEPARTMENT`);
console.log(result);
```

Create pool

Execute
using pool

Consistency amongst SDKs

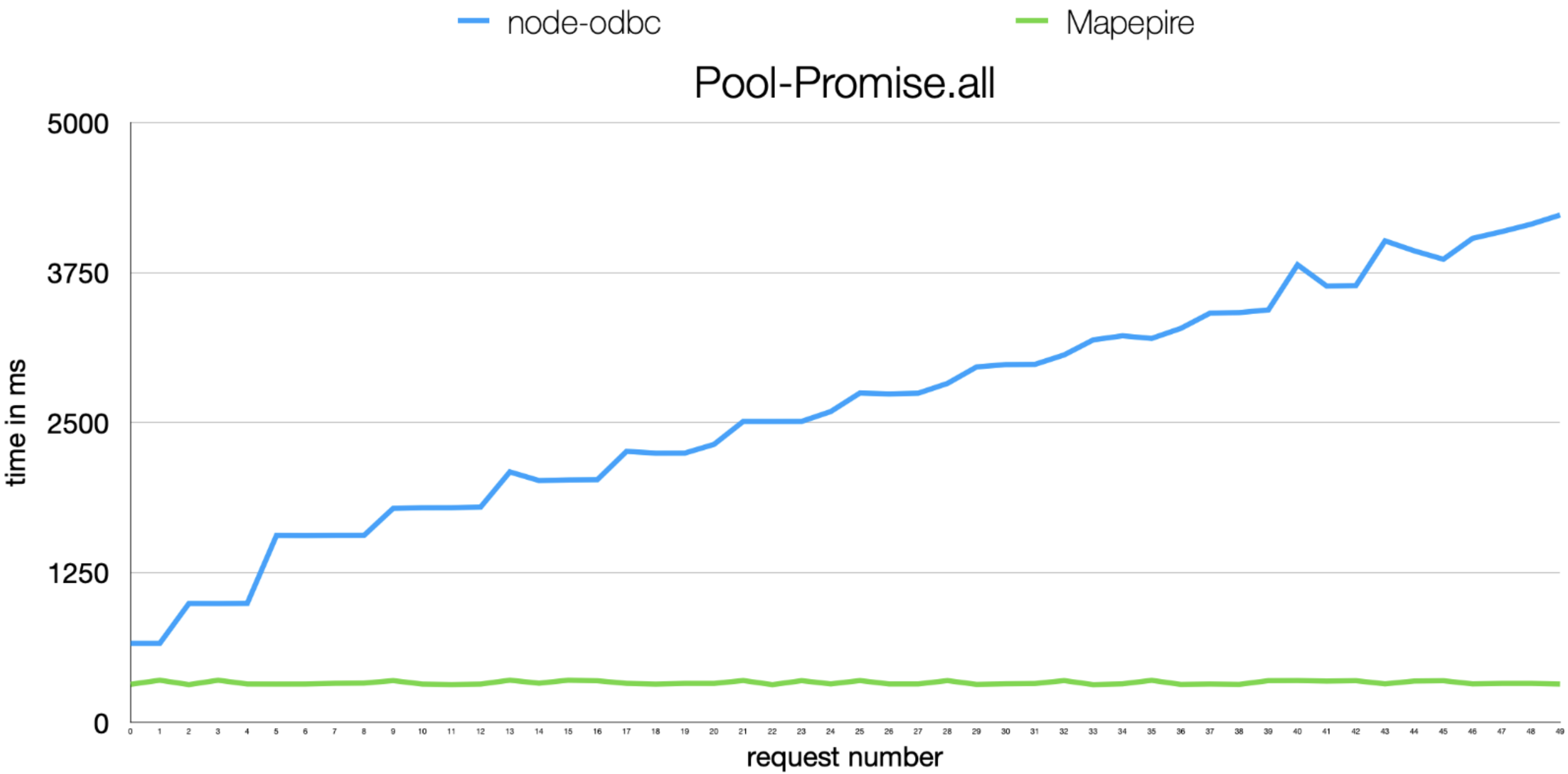
Node.js

```
// 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));
```

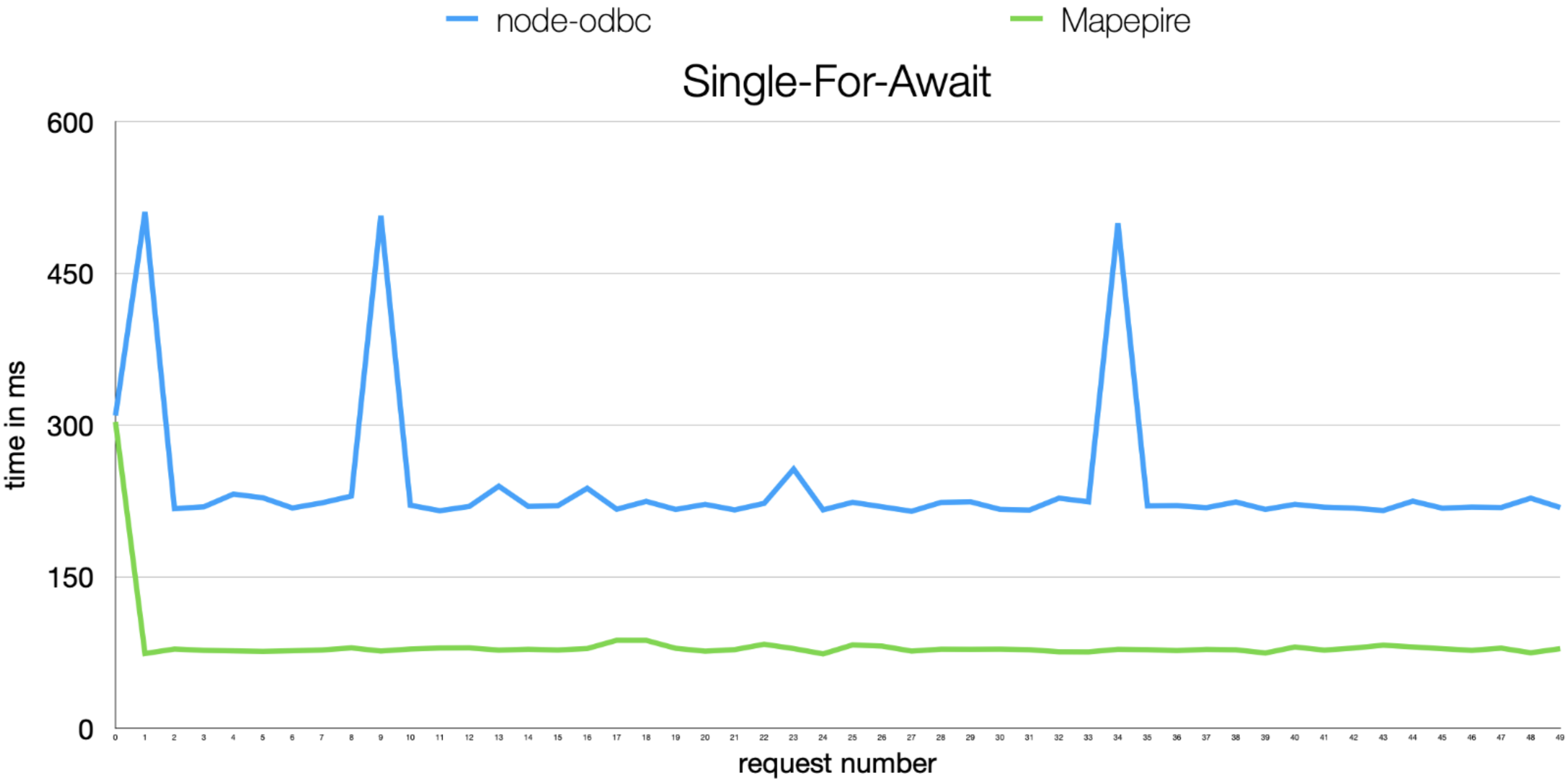
Java

```
// 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);
```

Let’s look at performance comparisons



More performance comparisons



What about encryption?

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

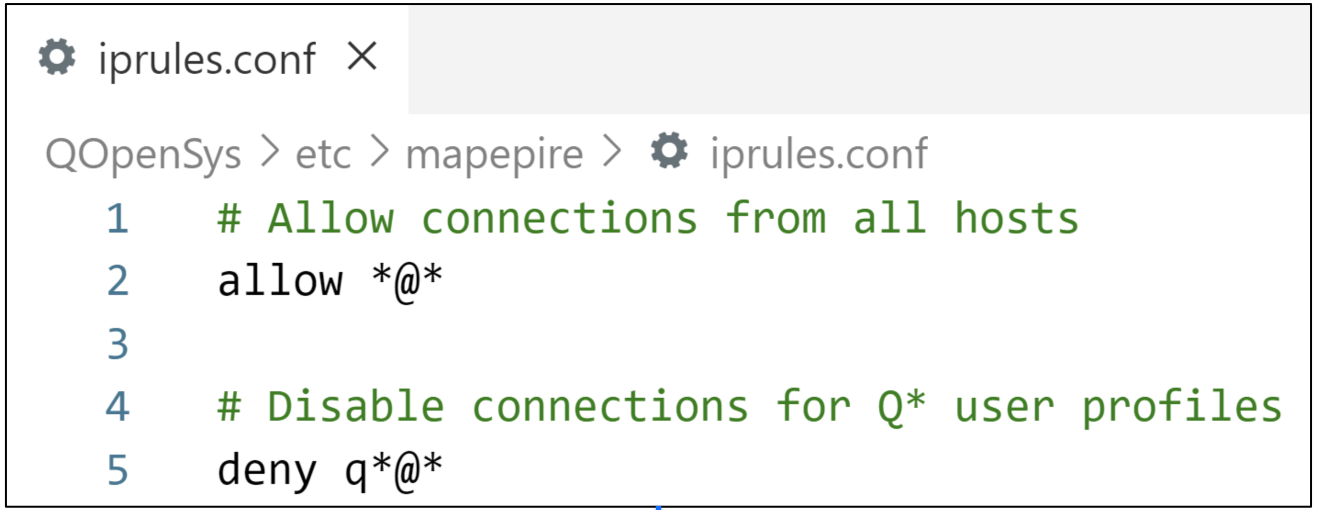
- Use Let's Encrypt (ex. generated by CertBot)
- Mapepire server will automatically use it as the server certificate
- Certificate must exist in the following location used by CertBot: `/etc/letsencrypt/live/<hostname>`

Option 3: Self-signed certificate

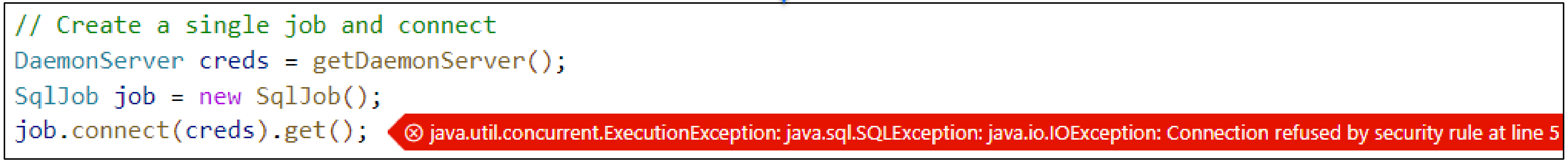
- If no certificate, the server automatically generates its own self-signed certificate

More security with user profile and IP filtering

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



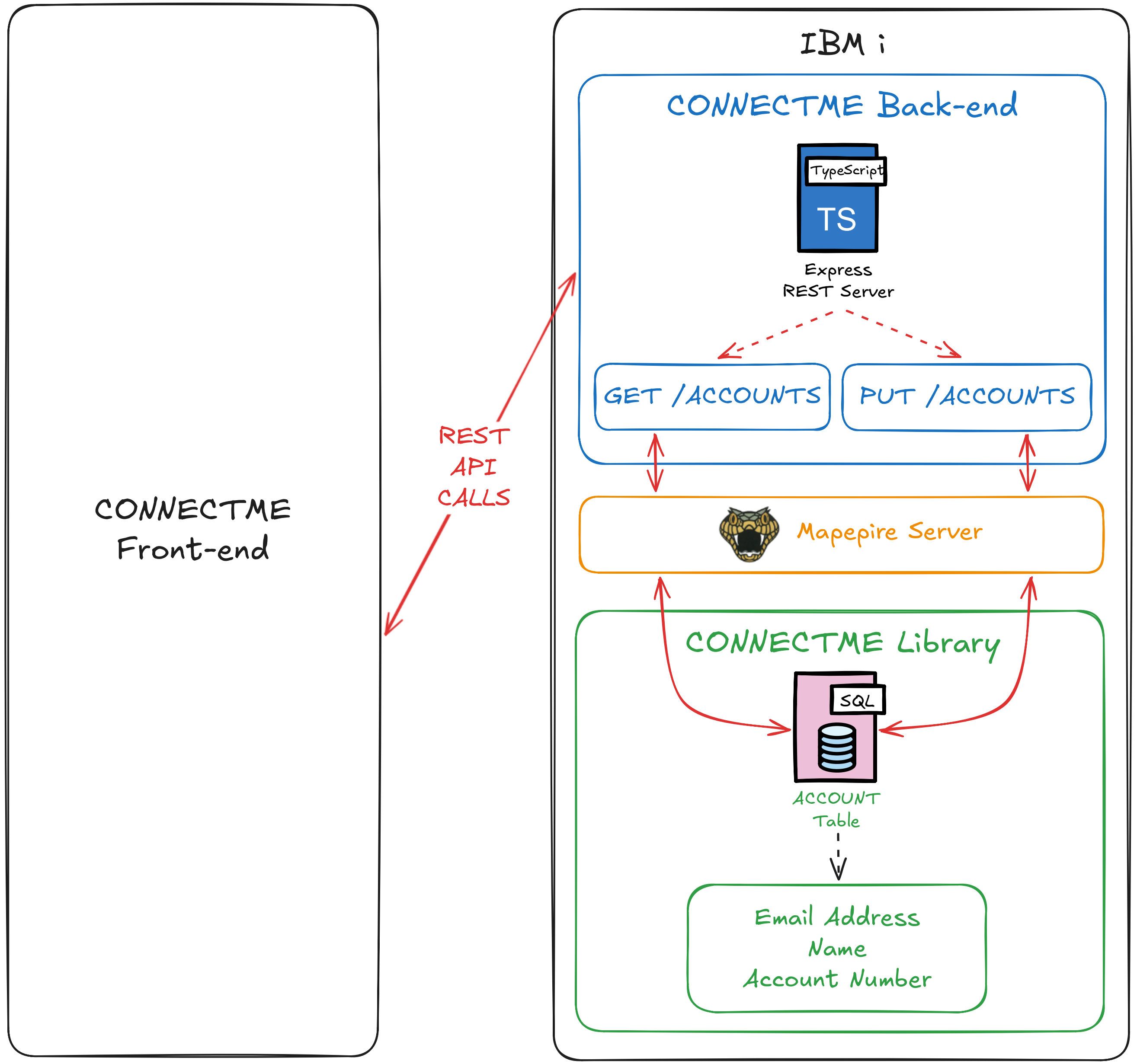
```
1    # Allow connections from all hosts
2    allow *@*
3
4    # Disable connections for Q* user profiles
5    deny q*@*
```



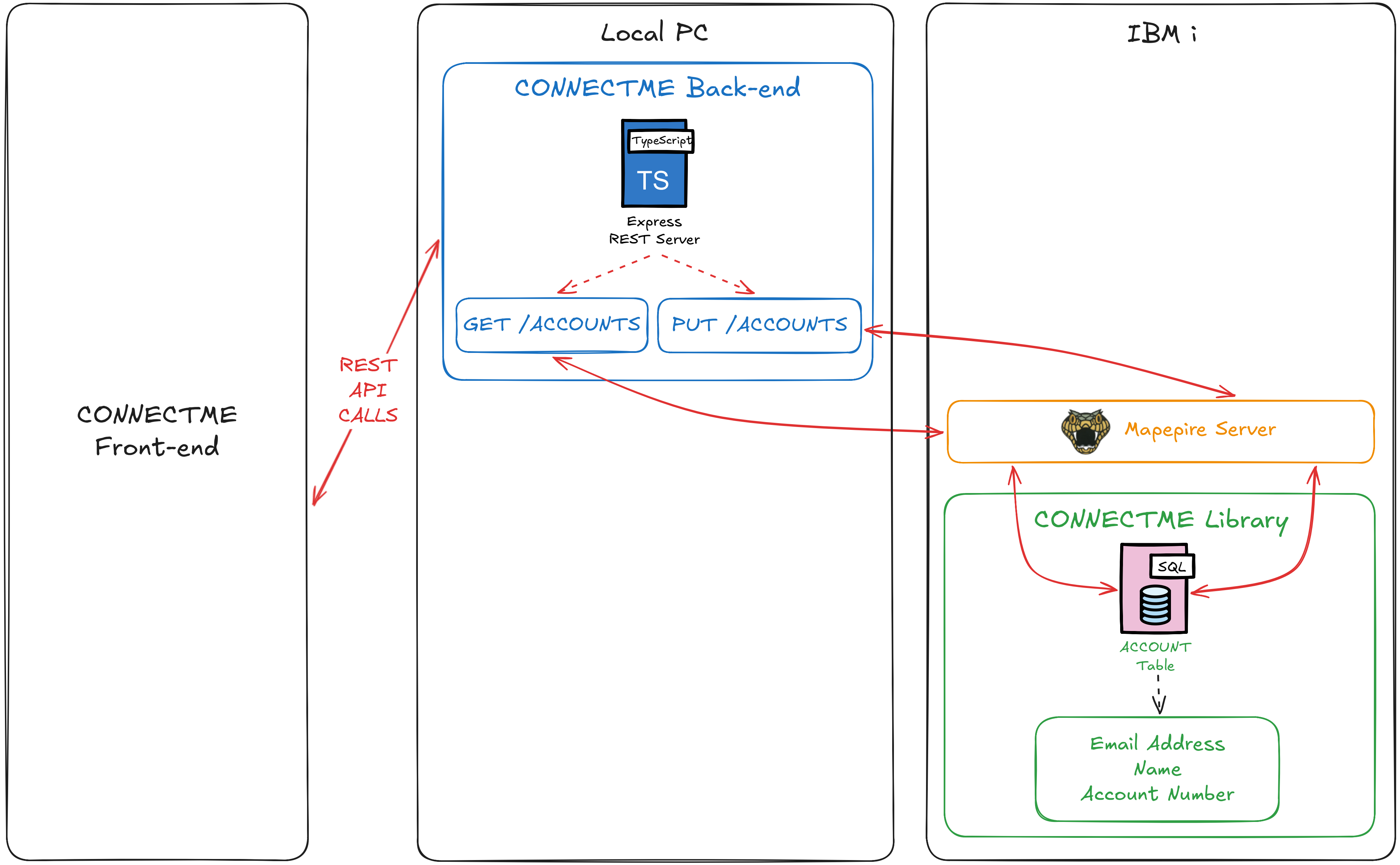
```
// Create a single job and connect
DaemonServer creds = getDaemonServer();
SqlJob job = new SqlJob();
job.connect(creds).get();
```

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

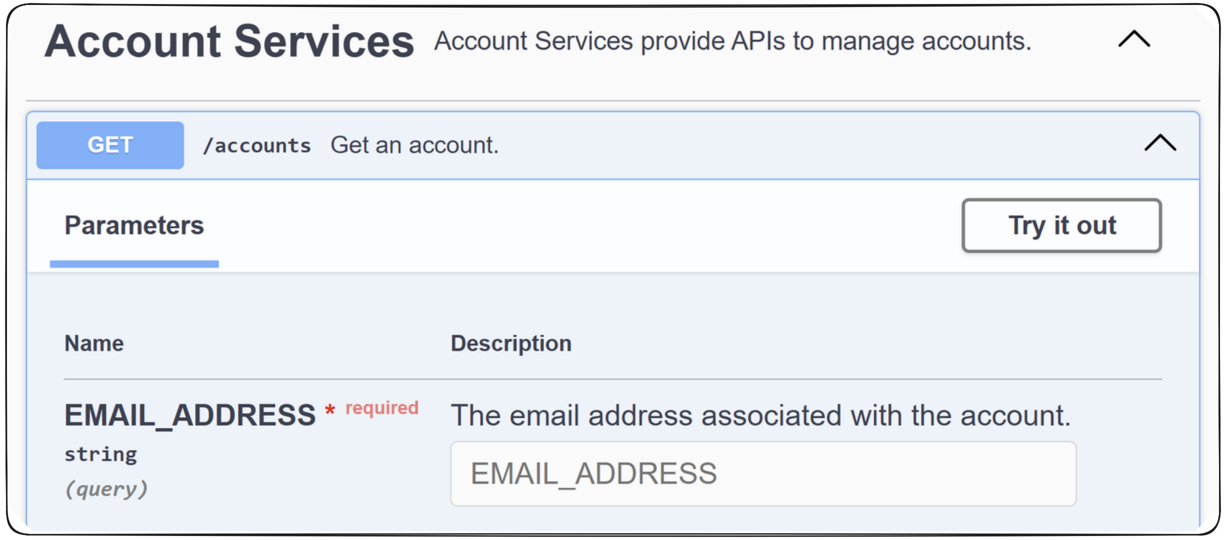
Let’s design a simple REST server using Mapepire



No platform specific driver...so the back-end can run anywhere!



Let’s start building the back-end



Account Services

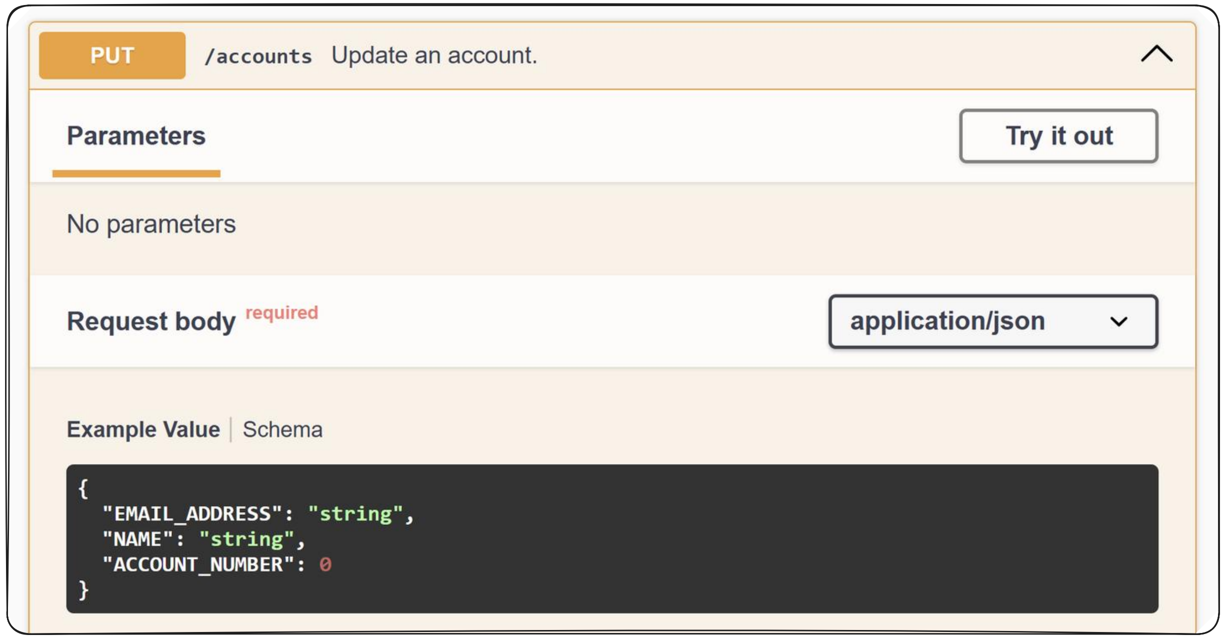
Account Services provide APIs to manage accounts.

GET `/accounts` Get an account.

Parameters

Try it out

Name	Description
EMAIL_ADDRESS * required `string` *(query)*	The email address associated with the account. EMAIL_ADDRESS



PUT `/accounts` Update an account.

Parameters

Try it out

No parameters

Request body required

application/json

Example Value | Schema

```
{
  "EMAIL_ADDRESS": "string",
  "NAME": "string",
  "ACCOUNT_NUMBER": 0
}
```

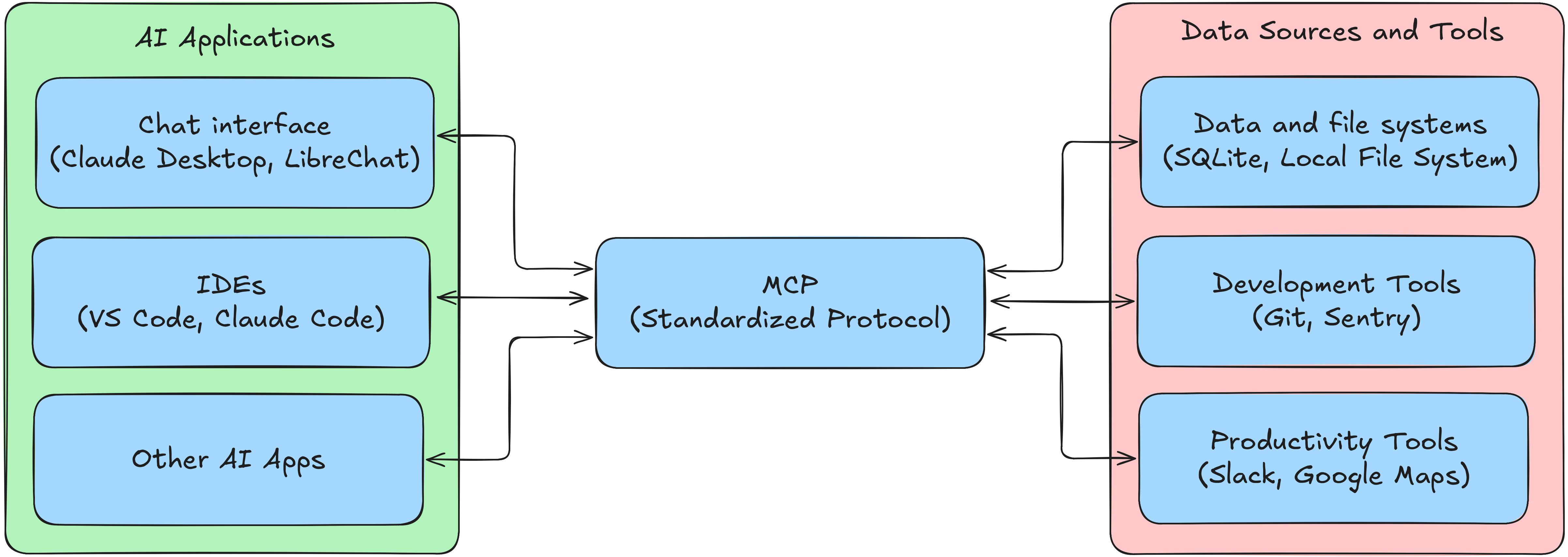
Mapepire documentation

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

Building an MCP Server

What is MCP?

MCP like a USB-C port for AI applications



Participants

MCP Host

The AI application that coordinates and manages one or multiple MCP clients

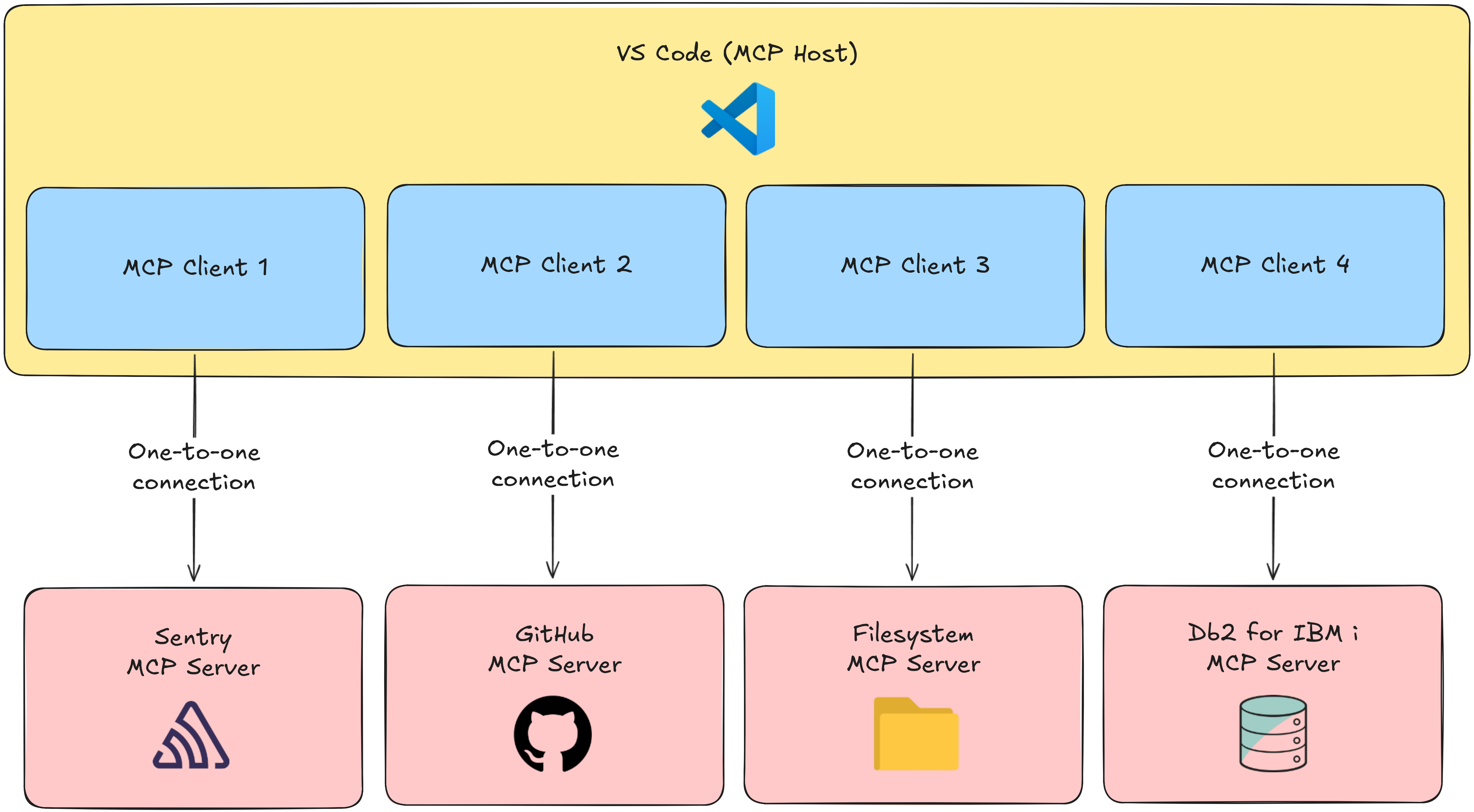
MCP Client

A component that maintains a connection to an MCP server and obtains context from an MCP server for the MCP host to use

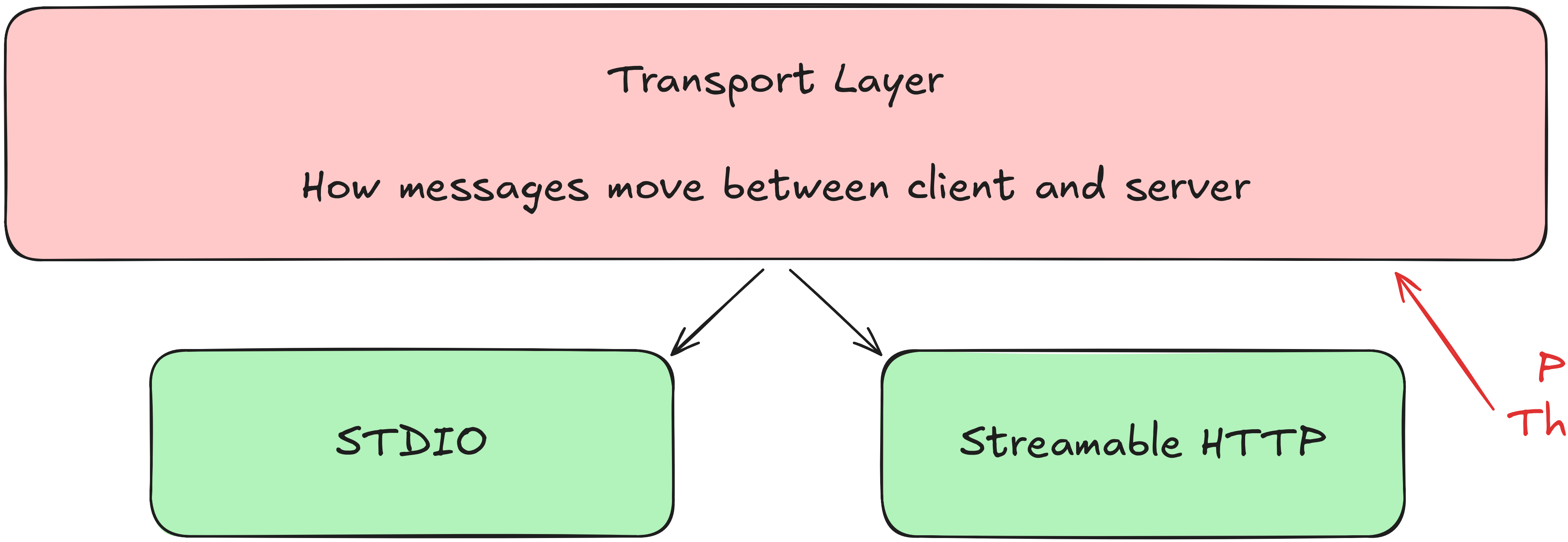
MCP Server

A program that provides context to MCP clients

Example participants



Transport layer



Transport Layer

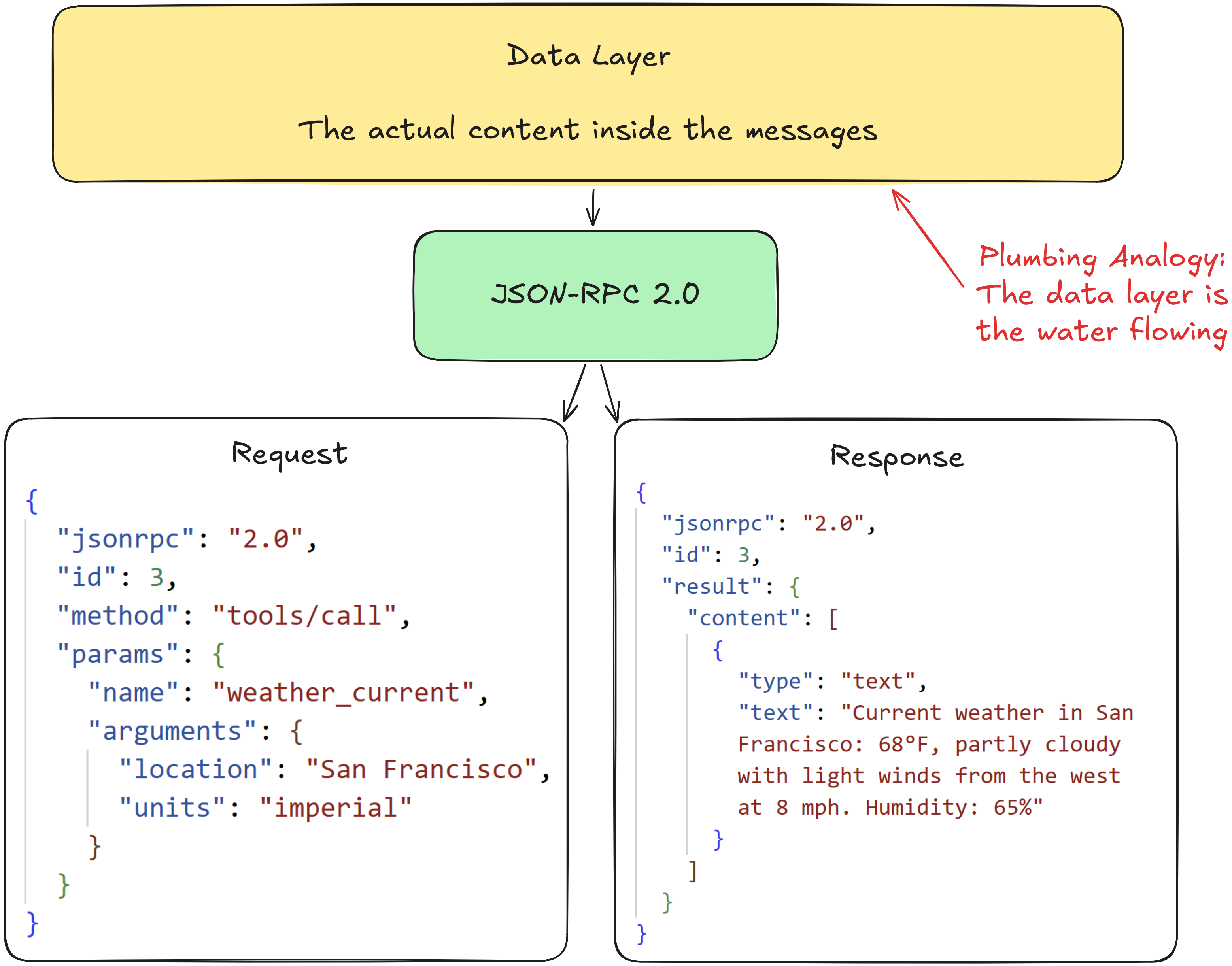
How messages move between client and server

STDIO

Streamable HTTP

Plumbing Analogy:
The transport layer
is the pipe

Data layer



Data Layer

The actual content inside the messages

JSON-RPC 2.0

Plumbing Analogy: The data layer is the water flowing

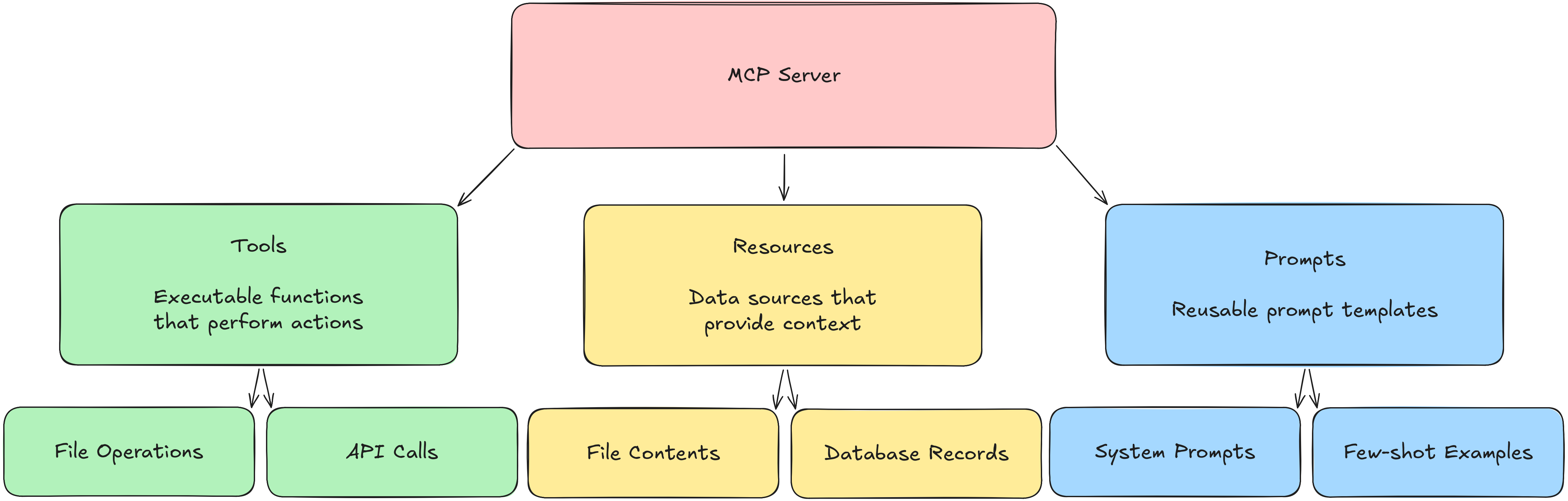
Request

```
{
  "jsonrpc": "2.0",
  "id": 3,
  "method": "tools/call",
  "params": {
    "name": "weather_current",
    "arguments": {
      "location": "San Francisco",
      "units": "imperial"
    }
  }
}
```

Response

```
{
  "jsonrpc": "2.0",
  "id": 3,
  "result": {
    "content": [
      {
        "type": "text",
        "text": "Current weather in San
        Francisco: 68°F, partly cloudy
        with light winds from the west
        at 8 mph. Humidity: 65%"
      }
    ]
  }
}
```

Primitives



MCP Server

Tools

Executable functions that perform actions

Resources

Data sources that provide context

Prompts

Reusable prompt templates

File Operations

API Calls

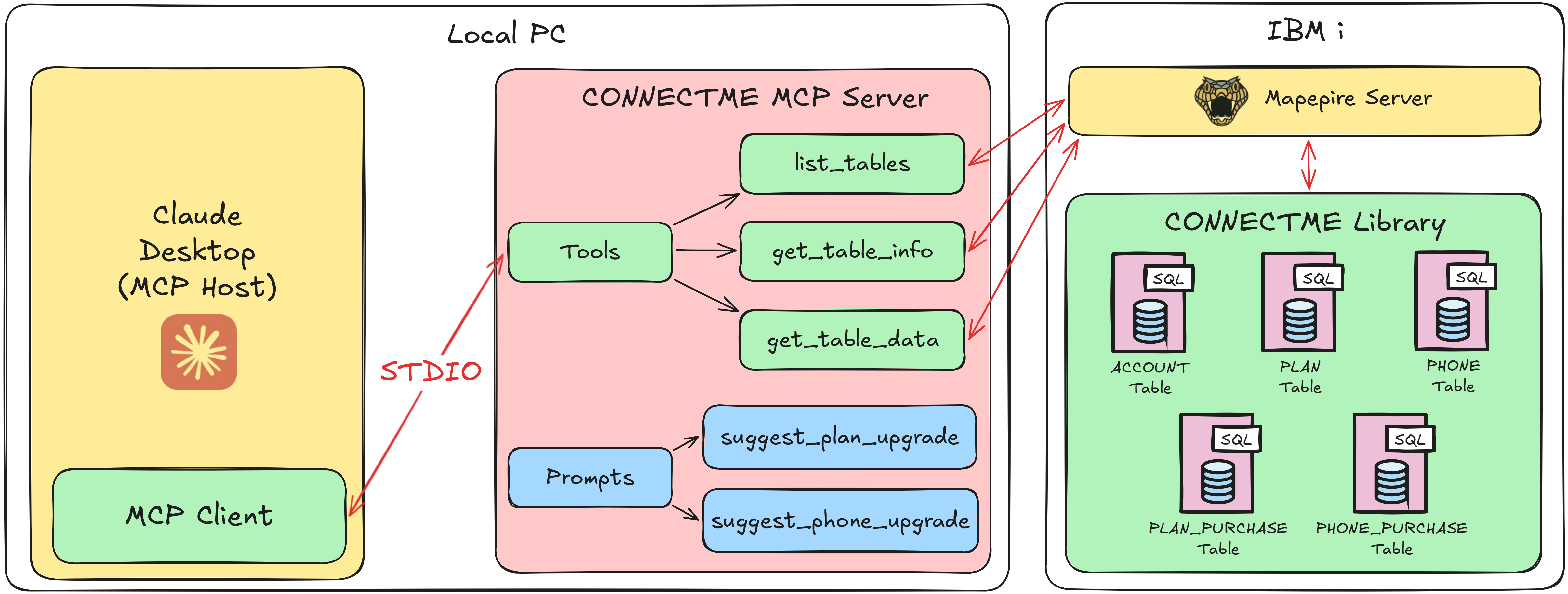
File Contents

Database Records

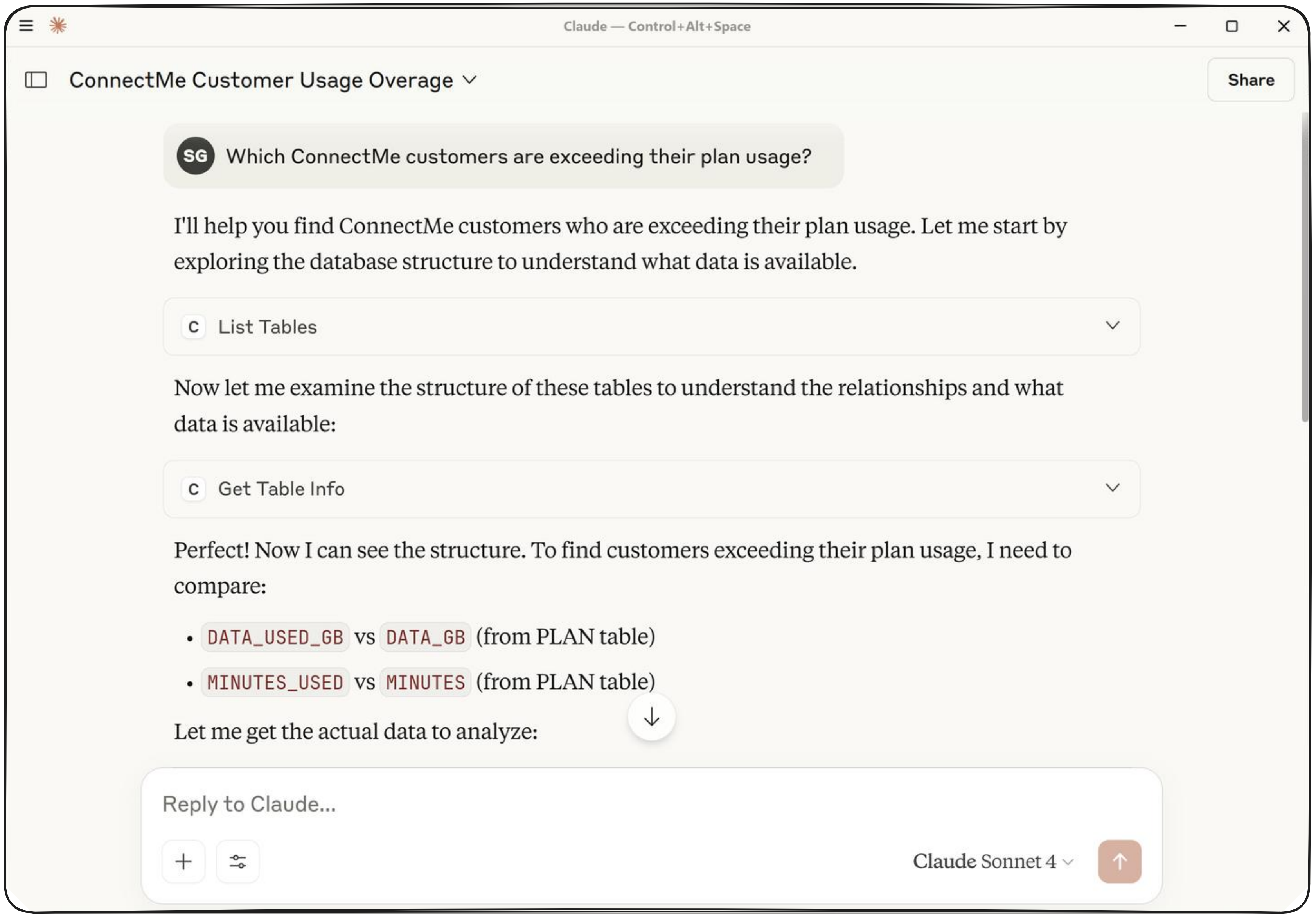
System Prompts

Few-shot Examples

Let’s design an MCP server



Let’s start building the MCP server



SG Which ConnectMe customers are exceeding their plan usage?

I'll help you find ConnectMe customers who are exceeding their plan usage. Let me start by exploring the database structure to understand what data is available.

Now let me examine the structure of these tables to understand the relationships and what data is available:

Perfect! Now I can see the structure. To find customers exceeding their plan usage, I need to compare:

- `DATA_USED_GB` vs `DATA_GB` (from PLAN table)
- `MINUTES_USED` vs `MINUTES` (from PLAN table)

Let me get the actual data to analyze:

Let’s try some prompts

"What phones do we sell at ConnectMe?"

"How many connectme accounts are open?"

"Do all 8 accounts have a phone with us?"

"What phone would you suggest to Frank? Consider what phone other customers are using who have the same plan."

"Which ConnectMe customers are exceeding their plan usage?"

suggest_plan_upgrade + Basic plan

suggest_plan_upgrade + "Keep your answer brief"

MCP documentation

https://modelcontextprotocol.io/docs/getting-started/intro

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
MCP	
Documentation	https://modelcontextprotocol.io/docs/getting-started/intro

Sanjula Ganepola - Node.js In Action - Intro to Mapepire and Building an MCP Server

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 i