#IDUGDb2

A RESTful Approach to Using Db2

George Baklarz

IBM

Session code: D4

Tuesday, 17 November, 15:00 - 15:50

Platform: Db2

Additional Thanks:
Peter Kohlmann
Dan Snoddy

Safe Harbor Statement

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON CURRENT THINKING REGARDING TRENDS AND DIRECTIONS, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. FUNCTION DESCRIBED HEREIN MY NEVER BE DELIVERED BY I BM. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS AND/OR SOFTWARE.

Agenda

- What is RESTful and Why should I care?
- A Beginner's Guide to RESTful Calls
- The Current RESTful Approaches for Db2
- Implementing RESTful with Db2 11.5.4
- Labs and Resources to help you out

What is RESTful?

What is RESTful?

- **Representational state transfer** (**REST**) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called *RESTful* Web services, provide interoperability between computer systems on the internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations.

Reference: https://en.wikipedia.org/wiki/Representational_state_transfer

Principles of RESTful Architecture

- Client-server architecture
- Statelessness
- Cacheability
- Layered system
- Code on demand (optional)
- Uniform interface

What is RESTful and Why Do I Care?

- Customers want to access Data from nontraditional platforms like Mobile platforms, from mobile apps (Android, Windows, iOS, etc.)
  - And in some cases want to publish access to data as well to 3rd parties
- Mobile and Cloud developers expect direct HTTP and JSON access to data without requiring Database Drivers
  - You want to install the Db2 drivers on your phone?
- In some environments, the use of a standards-based Query interface is preferred over native SQL-based queries

RESTful Methods

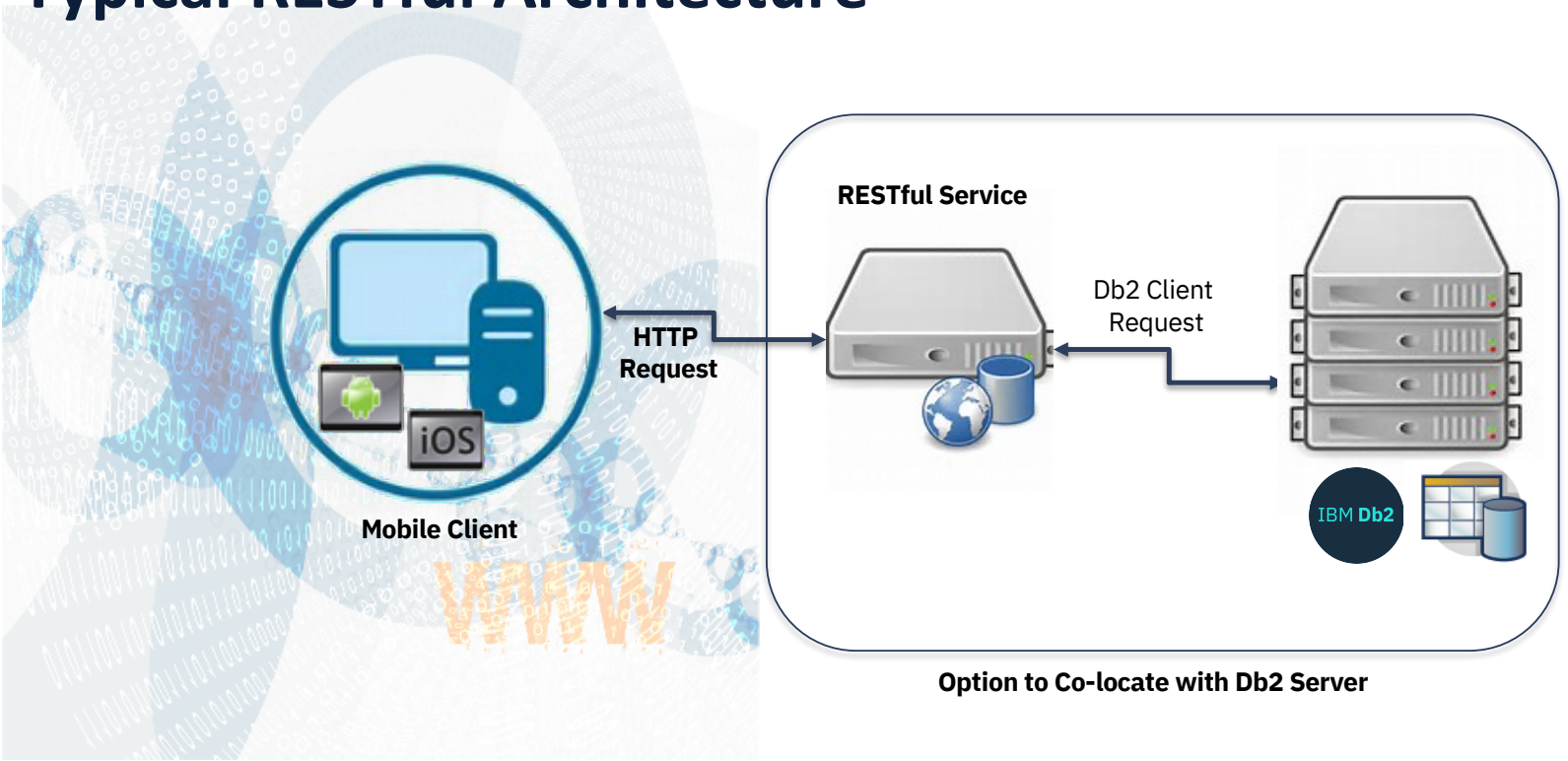
- POST
  - Invoke the operation provided by the resource
- GET
  - Retrieve a representation of the data in the response body
- PUT
  - Store the representation in the request body as the (new) state of the resource
- PATCH
  - Update some part of the resource's state
- DELETE
  - Delete the state of the resource

RESTful Choices

RESTful Approaches to Accessing Data

- Open Data Protocol (OData) Gateway
  - The IBM OData Gateway provides a RESTful service that uses the OData syntax to generate searches:
    `/EMPLOYEES?$select=LASTNAME,BONUS&$filter=contains(LASTNAME,'AA') and BONUS eq 1000`
- Db2 for Cloud Managed RESTful Service
  - The Db2oC RESTful service provides support for a majority of DDL and DML
  - Use POST and GET to execute SQL using native Db2 syntax
- Db2 RESTful Service with Data Management Console
  - Similar facilities to Db2 for Cloud
- Db2 Native RESTful Service
  - Installed as a component of Db2 11.5.4

Typical RESTful Architecture



A High-level Comparison of RESTful Approaches

	DDL	DML	Dynamic	App Endpoint	Security
OData Gateway	No	Limited	Limited	Limited	Limited
Db2 Cloud	Yes	Yes	Yes	No	Yes
Db2 DMC	Yes	Yes	Yes	No	Yes
Db2 11.5.4	Yes	Yes	Yes	Yes	Yes

DDL = Ability to create SQL objects

DML = Ability to run any type of SQL (Insert, Update, Select, Delete)

Dynamic = Modify the results of a RESTful endpoint

Fixed = Set RESTful Endpoint for applications

Security = Authentication Method

RESTful Availability

- IBM Data Server Gateway for OData V1.0
  - ibm.biz/idugd4-db2odata
- IBM Db2 on Cloud (Current)
  - ibm.biz/idugd4-db2cloud
- IBM Data Management Console 3.1
  - ibm.biz/idugd4-db2dmc
- IBM Db2 11.5.4 Restful Support
  - ibm.biz/idugd4-db2restful

Db2 RESTful

Db2 11.5.4 REST Service

- Released as part of Db2 11.5.4 in June as a standalone (Linux) container on IBM Cloud Container Registry
  - Can be installed on Docker independently of Db2
  - Also available as part of the Db2 Cartridge on OpenShift or Cloud Pak for Data
- Available on IBM Integrated Analytics System and Db2 Warehouse (local deployments) since February

Install the IBM Cloud Command Line Tools

- You need to have an IBM Cloud account (www.ibm.com/cloud)
- Go to Getting started with the IBM Cloud CLI and install the IBM Cloud command line tools
- **Note**: It installs a lot of tools!

- `Homebrew` (Mac only)
- `Git`
- `Docker`
- `Helm`
- `kubectl`
- `curl` (Linux™ only)
- IBM Cloud Functions plug-in
- IBM Cloud Object Storage plug-in
- IBM Cloud Container Registry plug-in
- IBM Cloud Kubernetes Service plug-in

Acquiring the Db2 REST Code

- Issue the following commands:
  - **`ibmcloud login -a cloud.ibm.com -sso`**
    - Authenticate using the one-time code and select your account
    - **Do not select** the region at this time!
  - **`ibmcloud cr region-set global`**
  - **`ibmcloud cr login`**
- Use Docker commands to pull and run the latest RESTful code
  - **`docker pull icr.io/ibm/hdm/db2rest:latest-amd64`**
  - **`docker run -it --net=host -e LICENSE=view --name=db2rest icr.io/ibm/hdm/db2rest:latest-amd64`**

Activating and Initializing REST Capability

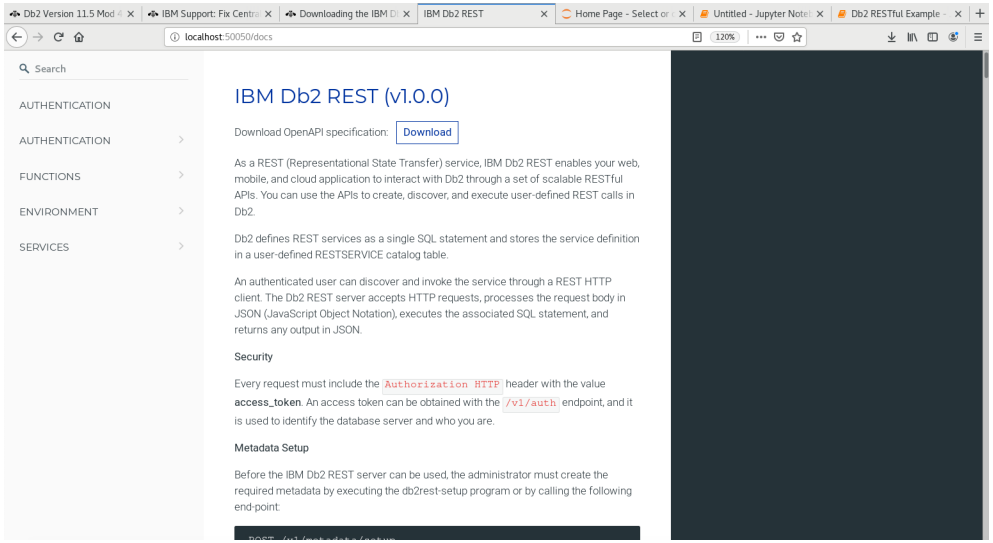
- To set up the RESTful server metadata, issue the following commands

```
docker exec containername /opt/ibm/dbrest/scripts/db2rest-stop.s
docker exec containername /opt/ibm/dbrest/scripts/db2rest-setup.sh
       hostname dbname port isSSLConnection schema username password
docker exec containername /opt/ibm/dbrest/scripts/db2rest-start.sh
```

- The setup step requires the following information
  - hostname – The IP address or name of the database server
  - dbname – Database name
  - port – Port number of the Db2 database
  - isSSLConnection – Y if using SSL for communication and N otherwise
  - schema – Schema name to used for RESTful metadata
  - username, password – Userid and password to access the database

Db2 REST: On-line Documentation

- Once the Db2 REST server is running, you can check that it is running is by viewing the on-line documentation
  - http://hostname:50050/docs



Managing the Db2 REST Server

- These scripts are used to operate the REST server

```
docker exec containername /opt/ibm/dbrest/scripts/scriptname
```

Script	Description
db2rest-start.sh	Starts the server and changes its status to ACTIVE.
db2rest-stop.sh	Stops the server and changes it status to INACTIVE. Any authentication tokens are automatically invalidated.
db2rest-restart.sh	Stops and restarts the server.
db2rest-version.sh	Returns the version information of a running server.
db2rest-status.sh	Checks the status (ACTIVE, INACTIVE, ERROR) of the server.

Db2 RESTful Development

REST Features

- Request authentication token
- Execute SQL statements
- Add/Modify Python User-defined Functions
- Create REST service end-points
- Update/delete REST end-points
- List and describe available end-points
- Execute and monitor job process
- Set service creation and execution permissions

Developing With REST

- Programming Support
  - You need the appropriate libraries in your favorite programming language to make RESTful calls (Almost all languages support RESTful!)
- REST Server Information
  - IP Address/Name
  - Port Number
  - Library Version
  - Service API Names
- Database Credentials
- Examples use Python Syntax

RESTful Call Structure

Server and Port

```
host = "http://172.16.210.132:50050"
```

Header with Handshake and Authorization Token

```
headers = {"authorization": f"{token}",
           "content-type": "application/json"}
```

Service Endpoint

```
service = "/v1/auth"
```

Body with Parameters

```
body = {"limit": 0}
```

RESTful Call Type

```
GET, POST, PUT, PATCH, DELETE
```

Final Call

```
response = requests.post(host+service,headers=headers,json=body)
```

Authentication for RESTful Calls

- Database Credentials Required for all Tasks
  - For running dynamic SQL
  - Creating/Maintaining/Running RESTful Services
  - Creating Python Functions
- Database Credentials
  - **dbHost** – IP address/name of the server
  - **dbName** – Database name
  - **dbPort** – Port Number (for database, not RESTful)
  - **isSSLConnection** – False for non-SSL, True for SSL
  - **username** – Userid to authenticate with
  - **password** – Password

```
db2id = {
        "dbHost": "172.16.210.132",
        "dbName": "SAMPLE",
        "dbPort": 50000,
        "isSSLConnection": False,
        "username": "db2inst1",
        "password": "db2inst1"
        }
```

Authentication and Access Tokens

- Everything revolves around Access Tokens
  - Access Tokens are used for all RESTful calls
  - An initial RESTful call uses database credentials to generate an access token
  - Subsequent database calls will use the Access token instead of userid/password
- To generate an access token you need to create a RESTful call that contains the following information:
  - Server and Port of the RESTful server
  - Header for Handshake information
  - Authentication Endpoint
  - Database Credentials

Authentication Arguments

- Server and Port

```
host = "http://172.16.210.132:50050"
```

Server could be different

- Header

```
headers = {"content-type": "application/json"}
```

Handshake

- Service Endpoint

```
API_Auth = "/v1/auth"
```

Version + Endpoint Service

- Body
  - **dbParms** – Database Connection Parameters
  - **expiryTime** – Amount of time the token is valid

```
body = {
       "dbParms": db2id,
       "expiryTime": "300m"
       }
```

Parameters

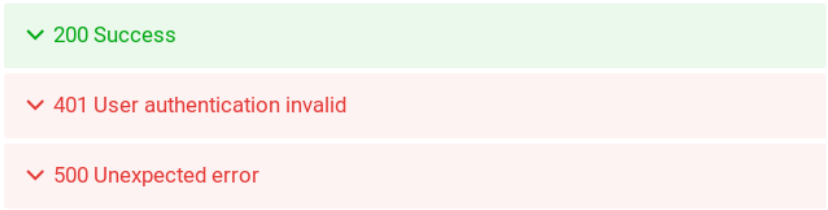
```
db2id = {
        "dbHost": "172.16.210.132",
        "dbName": "SAMPLE",
        "dbPort": 50000,
        "isSSLConnection": False,
        "username": "db2inst1",
        "password": "db2inst1"
        }
```

Authentication RESTful Call

```
# RESTful Host
RESTHost = http://172.16.210.132:50050

# Database Credentials
db2id = {
    "dbHost": "172.16.210.132",
    "dbName": "SAMPLE",
    "dbPort": 50000,
    "isSSLConnection": False,
    "username": "db2inst1",
    "password": "db2inst1"
}

# Service Endpoint
API_Auth = "/v1/auth"

# Header
headers = {"content-type": "application/json"}

# Body
body = {
  "dbParms": db2id,
  "expiryTime": "300m"
}

# RESTful Call
response = requests.post(RESTHost + API_auth, headers=headers, json=body)
```

Authentication Results

- RESTful service returns the result of the connection attempt



- Successful Connection returns a Token

```
response = requests.post(RESTHost + API_auth, headers=headers, json=body)
if (response.status_code == 200):
  token = response.json()["token"]
else:
  print(response.json()["errors"])
```

- The output will include any error messages returned from Db2

```
[{'code': 'authentication_failure', 'message': 'SQLDriverConnect: {08001} [IBM][CLI
Driver] SQL30082N  Security processing failed with reason "24" ("USERNAME AND/OR
PASSWORD INVALID").  SQLSTATE=08001\n'}]
```

REST Error Codes

- 200 Success
- 202 Accepted asynchronous request
- 204 Success
- 400 Bad request
- 401 User authentication invalid
- 403 User permissions invalid
- 404 The service was not found
- 500 Unexpected error

Execute Dynamic SQL

- The **execsql** service provides the ability to run any SQL statement
  - Includes all DML, DCL, and DDL statements
- Multiple statements can be placed into the body using a semicolon as the delimiter
- The **isQuery** option (True/False) indicates whether an answer set is to be returned
- The **sync** option (True/False) determines whether the RESTful call will wait for the results to be returned or if the program will poll for the results

Execute Dynamic SQL - Syntax

- Header

```
headers = {"authorization": f"{token}", "content-type": "application/json"}
```

- Service Endpoint

```
API_execsql = "v1/services/execsql"
```

- Body

```
body = {
  "isQuery": False,
  "sqlStatement": "CREATE TABLE A (X INT); INSERT INTO A VALUES 1,2,3,4;",
  "sync": True
}
```

- RESTful Call

```
response = requests.post(RESTHost + API_execsql, headers=headers, json=body)
```

Execute Dynamic SQL - Return Values

- The response variable will contain details of the SQL call
- For SQL without any answer sets, the **resultSet** will contain the amount of rows affected with any of the commands

```
{'jobStatus': 4,
'jobStatusDescription': 'Job is complete',
'resultSet': [{'rowsAffected': 1}],
'rowCount': 1}
```

- For SQL with answer sets, the data will be found in the **resultSet**:

```
{'resultSet': [
   {'BIRTHDATE': '1963-08-24T00:00:00Z','BONUS': 1000,'COMM': 4220,
    'EDLEVEL': 18,'EMPNO': '000010','FIRSTNME': 'CHRISTINE',
    'HIREDATE': '1995-01-01T00:00:00Z','JOB': 'PRES    ','LASTNAME': 'HAAS',
    'MIDINIT': 'I','PHONENO': '3978','SALARY': 152750,'SEX': 'F','WORKDEPT': 'A00'}
    ]
}
```

Execute Dynamic SQL – Polling of Values

- To prevent an application from blocking on a long-running SQL statement, the **sync** setting can be set to **False**
- The RESTful call will return immediately and provide a job number that can be used to retrieve the answer set
- The response will contain the ID of the job:

```
{'id': 'c4adf010-6796-45b4-9bab-a354bdf99a3b'}
```

- To retrieve the results you would use a **GET** call using this **ID** value

Execute Dynamic SQL – Retrieving Result Sets

- Header

```
headers = {"authorization": f"{token}", "content-type": "application/json"}
```

- Service Endpoint

```
API_get = "v1/services/"
```

- Body

```
body = {"limit": 0}
```

- RESTful Call

```
response = requests.get(RESTHost + API_get + job_id, headers=headers,json=body)
```

- Results found in **resultset**

```
{'resultSet': [{'BIRTHDATE': '1963-08-24T00:00:00Z', 'BONUS': 1000, 'COMM': 4220,
 'EDLEVEL': 18, 'EMPNO': '000010', 'FIRSTNME': 'CHRISTINE',
 'HIREDATE': '1995-01-01T00:00:00Z', 'JOB': 'PRES    ', 'LASTNAME': 'HAAS',
 'MIDINIT': 'I', 'PHONENO': '3978', 'SALARY': 152750, 'SEX': 'F', 'WORKDEPT': 'A00'}]
```

Execute Dynamic SQL – Parameters

- SQL can include parameter markers by using a "**?**" where a variable is required

```
SELECT * FROM EMPLOYEE WHERE EMPNO = ?
```

- Add parameters to the body and index them starting at one

```
body = {
  "isQuery": True,
  "parameters" : {
      "1" : "000010"
  },
 "sqlStatement": "SELECT * FROM EMPLOYEE WHERE EMPNO=?",
 "sync": True
}
```

- Especially useful when dealing with quoted strings

Creating a RESTful Service

- Rather than use a generic **execsql** service, a developer can create a specific RESTful service for use in an application
- The settings are similar to running dynamic SQL
  - **isQuery** – (True/False) Does the SQL return an answer set
  - **parameters** – A list of parameters required by the SQL
  - **schema** – Optional schema if not supplied in the SQL
  - **serviceDescription** – A description of the service (for querying services)
  - **serviceName** – The name of the service (must be unique)
  - **sqlStatement** – The SQL to execute (SELECT, INSERT, UPDATE, DELETE)
  - **version** – The version of the service

Creating a RESTful Service – Body Syntax

```
body = {
       "isQuery": true,
       "parameters": [
         {
         "datatype": "VARCHAR(8)",
         "name": "@empno"
         }
       ],
       "schema": "DB2INST1",
       "serviceDescription": "Return the last name of the employee",
       "serviceName": "getemployee",
       "sqlStatement": "SELECT LASTNAME FROM EMPLOYEE WHERE EMPNO=@empno",
       "version": "1"
}
```

Creating a RESTful Service - Parameters

- The SQL statement can contain parameters
  - Parameters are indicated with an **@** sign in front of the value

    EMPNO=@empno
- To supply the values to the SQL statement, use the **parameters** field

```
"parameters": [
      {
      "datatype": "VARCHAR(8)",
      "name": "@empno"
      }
],…
```

- The datatype must map to a Db2 datatype and the name must include the **@** sign

Calling a RESTful Service

- The RESTful endpoint is a combination of the RESTful service name and the version

```
API_runrest = "/v1/services/getemployee/1"
```

- The body settings are similar to running dynamic SQL
  - **parameters** – A list of parameters required by the SQL
  - **sync** – True (wait for the results) or False (poll for the results)
- The parameter list must use the same variables names as in the SQL

```
body = {
  "parameters": {
  "@empno": "000010"
  },
  "sync": True
}
```

Db2 RESTful Management

RESTful Service Maintenance

- Add/Remove Access
  - **PUT /v1/services/grant** – Grants permission to execute a REST service
  - **DELETE /v1/services/grant** – Revokes permission to execute a REST service
- Services Utilities
  - **GET /v1/services/monitor** – Monitor the executing services
  - **PUT /v1/services/stop/id** – Stop a job
  - **GET /v1/services/restname/version** – Describes the details of the service
  - **DELETE /v1/services/restname/version** – Deletes the service
  - **PATCH /v1/services/restname/version** – Updates the description or SQL associated with a service

RESTful Server Maintenance

- **GET /v1/logs**
  - Download the REST server logs as a zip file
- **PUT /v1/metadata/grant**
  - Grants permission to create REST services
- **DELETE /v1/metadata/grant**
  - Revokes permission to create REST services
- **POST /v1/metadata/setup**
  - Setup the metadata for the RESTful server
- **GET /v1/version**
  - Gets the RESTful server version

Python Library Functions

- **GET /v1/functions**
  - Gets the Python user-defined functions
- **POST /v1/functions**
  - Creates a new Python user-defined function
- **POST /v1/functions/file**
  - Uploads a Python file
- **GET /v1/functions/schema/name**
  - Describes the details of a Python user-defined function
- **DELETE /v1/functions/schema/name**
  - Deletes a Python user-defined function

Resources

Resources

- IBM Data Management Console 3.1
  - ibm.biz/idugd4-db2dmc
- IBM Db2 11.5.4 Restful Support
  - ibm.biz/idugd4-db2restful
- IBM Db2 on OpenShift Hands on Lab
  - www.click2containerize.com


IDUG
Leading the Db2 User
Community since 1988
George Baklarz
IBM
baklarz@ca.ibm.com
#IDUGDb2