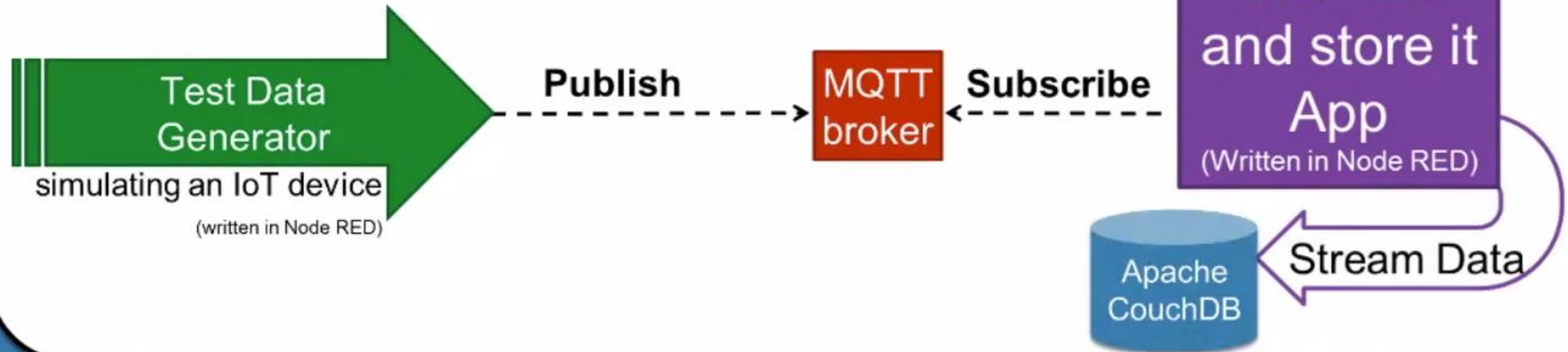


TEST DATA GENERATION

End-to-end scenario

IoT Platform



FLOW I

The screenshot shows the Node-RED web interface. On the left is the 'input' and 'output' node palette. The main workspace shows a flow editor with a text editor open, displaying a JSON object. The text editor is titled 'flow1.js' and contains the following JSON:

```
1 [{ "id": "435b3f9c.11764", "type": "ibmiot  
in", "z": "38f93775.f14ba8", "authentication": "boundService", "apiKey": "", "i  
nputType": "evt", "deviceId": "", "applicationId": "", "deviceType": "+", "event  
Type": "+", "commandType": "", "format": "json", "name": "IBM  
IoT", "service": "registered", "allDevices": "", "allApplications": "", "allDev  
iceTypes": true, "allEvents": true, "allCommands": "", "allFormats": "", "qos": 0  
, "x": 181.5, "y": 214.25, "wires": [[ "13a9bb35.c3a655" ] ] },  
{ "id": "13a9bb35.c3a655", "type": "cloudant  
out", "z": "38f93775.f14ba8", "name": "", "cloudant": "", "database": "washing",  
"service": "noderedrkie-  
cloudantNoSQLDB", "payonly": true, "operation": "insert", "x": 484.5, "y": 308.2  
5, "wires": [ ] } ] }
```

An arrow points from the text editor to the right, indicating it can be opened in a text editor.

Open in a text editor

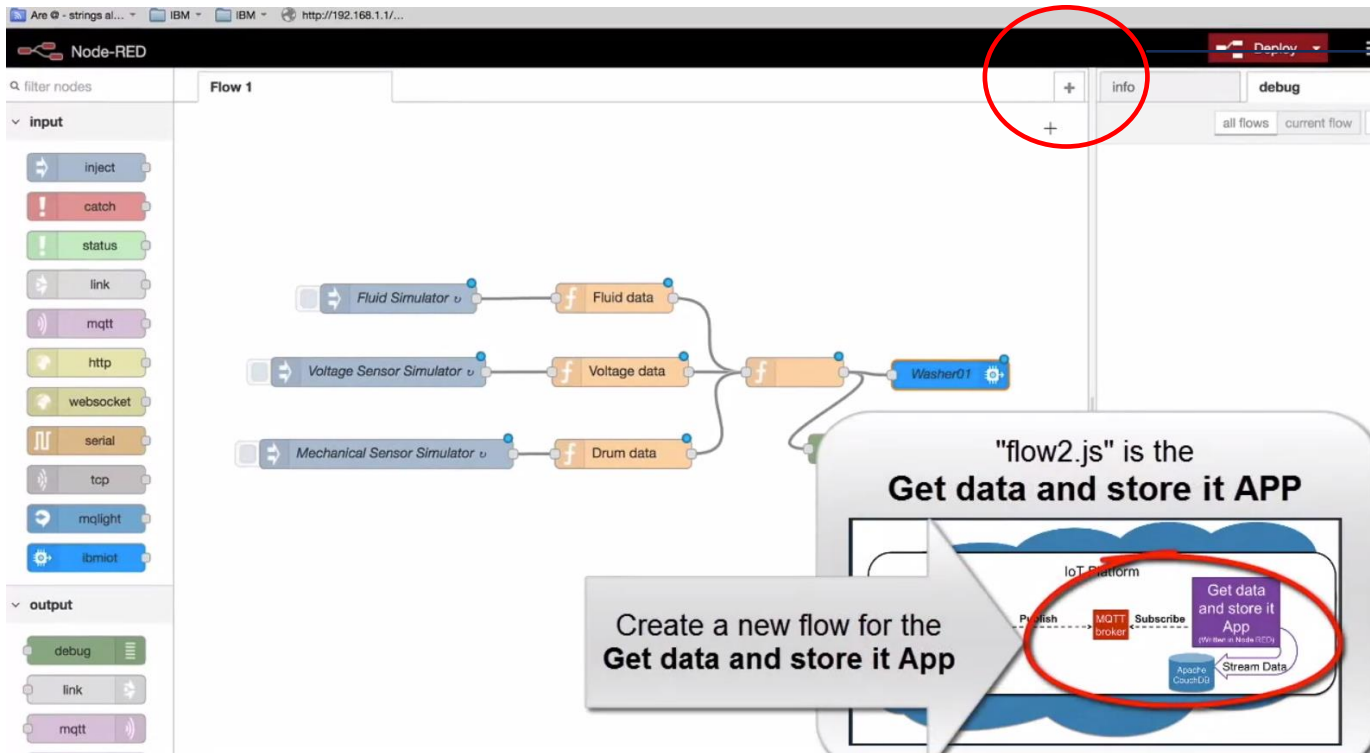
Past here:

The screenshot shows the Node-RED web interface. On the left is the 'input' and 'output' node palette. The main workspace shows a flow editor with a text editor open, displaying a JSON object. The text editor is titled 'flow1.js' and contains the following JSON:

```
1 [{ "id": "435b3f9c.11764", "type": "ibmiot  
in", "z": "38f93775.f14ba8", "authentication": "boundService", "apiKey": "", "i  
nputType": "evt", "deviceId": "", "applicationId": "", "deviceType": "+", "event  
Type": "+", "commandType": "", "format": "json", "name": "IBM  
IoT", "service": "registered", "allDevices": "", "allApplications": "", "allDev  
iceTypes": true, "allEvents": true, "allCommands": "", "allFormats": "", "qos": 0  
, "x": 181.5, "y": 214.25, "wires": [[ "13a9bb35.c3a655" ] ] },  
{ "id": "13a9bb35.c3a655", "type": "cloudant  
out", "z": "38f93775.f14ba8", "name": "", "cloudant": "", "database": "washing",  
"service": "noderedrkie-  
cloudantNoSQLDB", "payonly": true, "operation": "insert", "x": 484.5, "y": 308.2  
5, "wires": [ ] } ] }
```

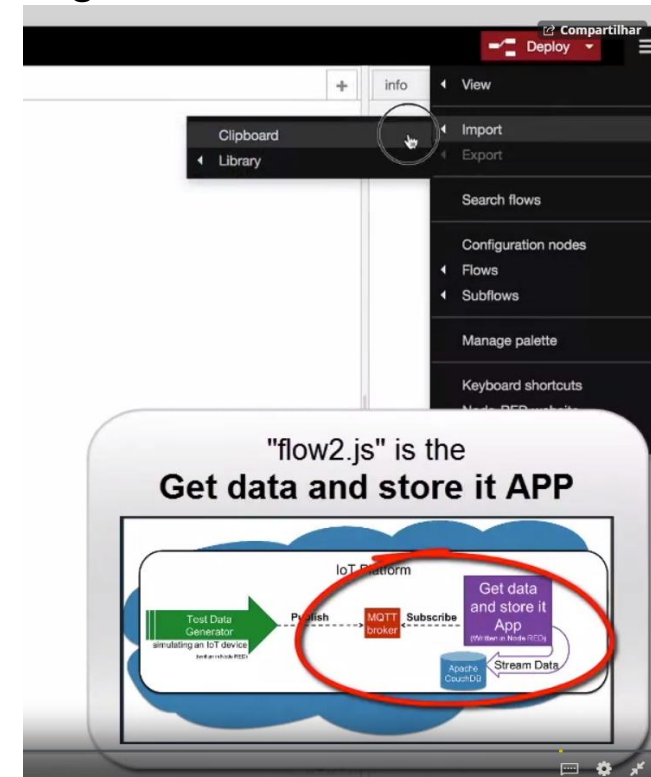
An arrow points from the text editor to the right, indicating it can be opened in a text editor.

FLOW 2

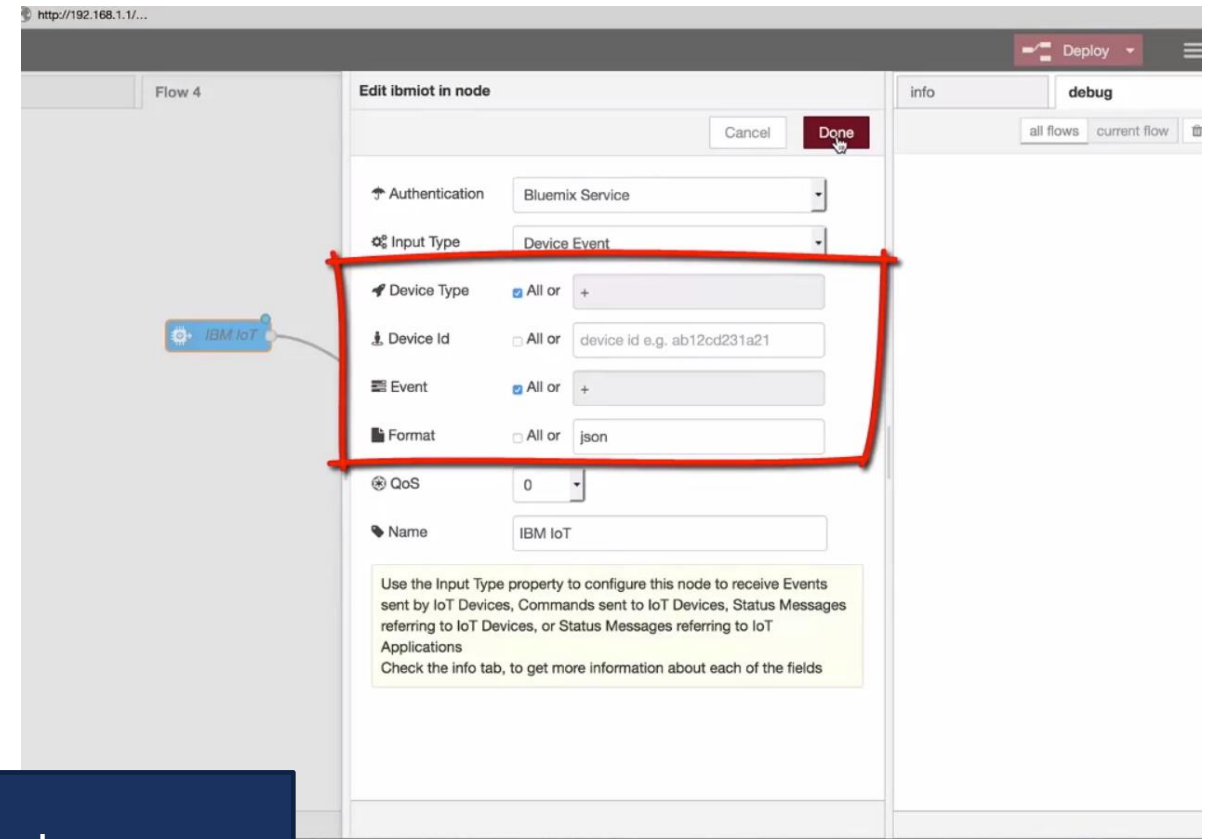
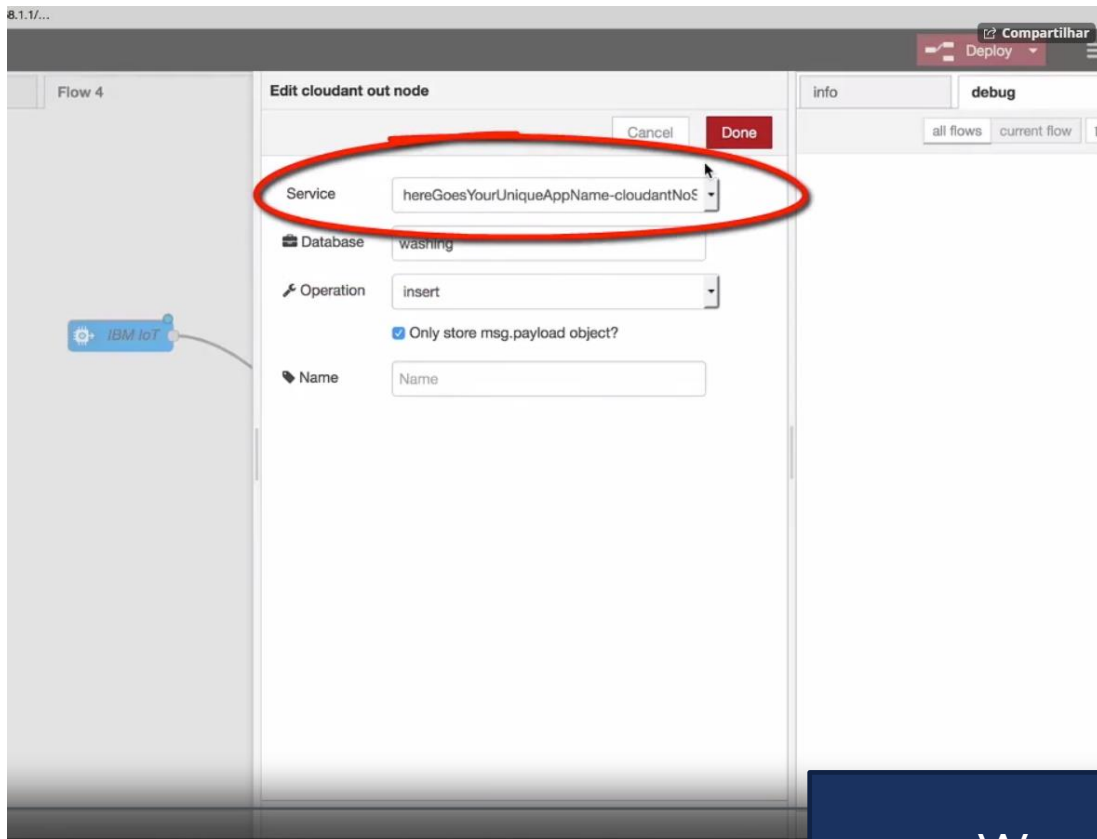


Click here to create a new panel for a new flow

Again:



IBM IOT AND WASHING (FLOW2)



We are ready to
Generate test data!

CLICK ON MSG.PAYLOAD

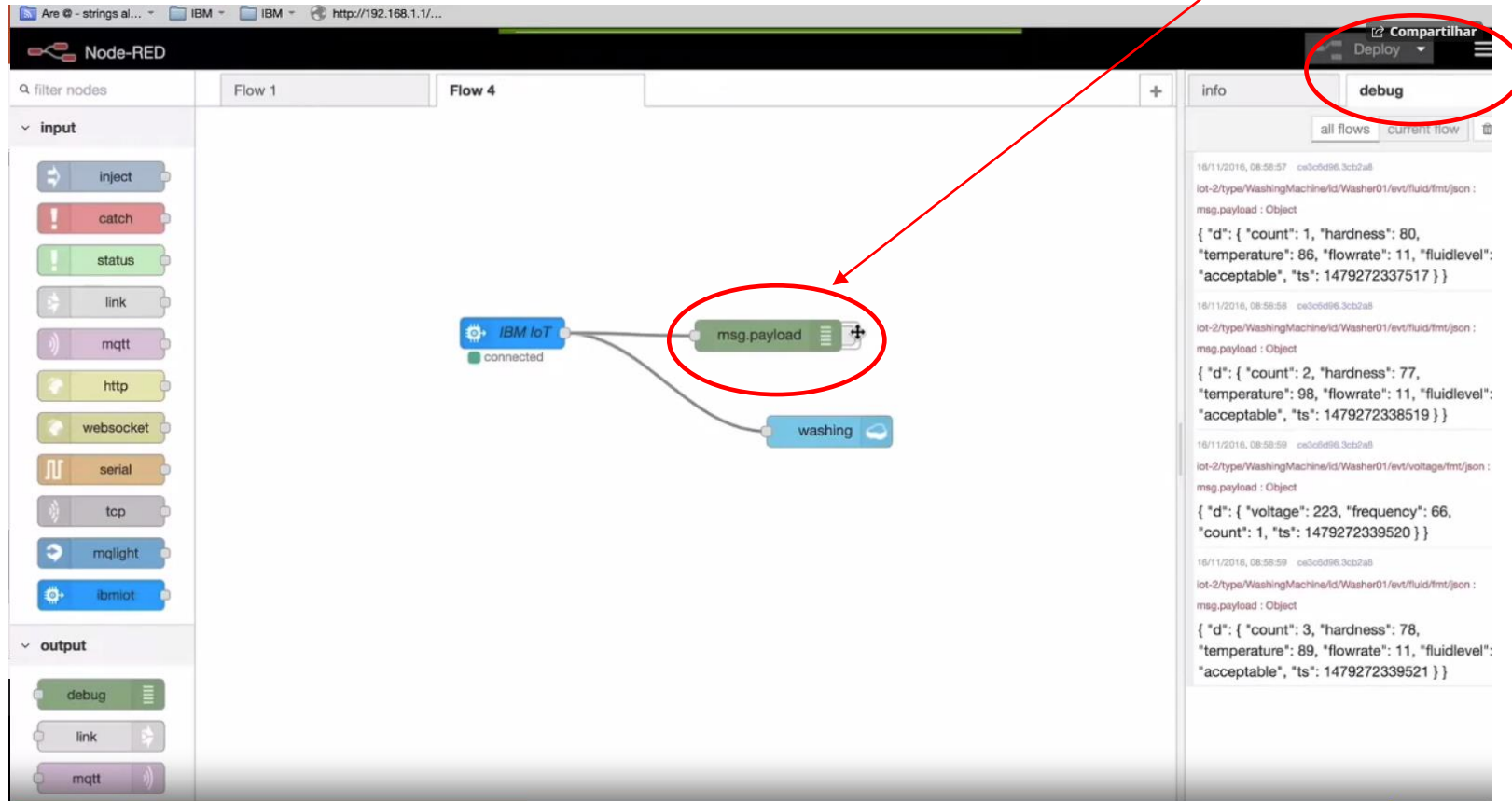
The screenshot shows the Node-RED web interface. On the left, there are panels for 'input' and 'output' nodes. The 'input' panel includes nodes like inject, catch, status, link, mqtt, http, websocket, serial, tcp, mqlight, and ibmiot. The 'output' panel includes debug, link, and mqtt. In the center, a flow is visible with three input nodes: 'Fluid Simulator', 'Voltage Sensor Simulator', and 'Mechanical Sensor Simulator'. Each is connected to a function node labeled 'Fluid data', 'Voltage data', and 'Drum data' respectively. These three function nodes are connected to a single function node, which is then connected to a 'Washer01' node. A red circle highlights the 'msg.payload' node in the flow. A red arrow points from the 'msg.payload' node to the 'debug' tab on the right. The 'debug' tab shows a list of messages with their payloads. The first message is an object with properties: 'count', 'hardness', 'temperature', 'flowrate', 'fluidlevel', 'acceptable', and 'ts'. The second message is an object with properties: 'voltage', 'frequency', 'count', and 'ts'. The third message is an object with properties: 'count', 'hardness', 'temperature', 'flowrate', 'fluidlevel', 'acceptable', and 'ts'. The fourth message is an object with properties: 'speed', 'count', and 'ts'. The fifth message is an object with properties: 'count', 'hardness', 'temperature', 'flowrate', 'fluidlevel', 'acceptable', and 'ts'.

Once we activate the debug node, we will see test data generated by the test data generator. We'll have a look at this later. But basically, it is important to notice that this particular flow normally doesn't run in the cloud, but on an IoT device or gateway reading raw sensor data directly from the built-in sensors and transmitting those directly to the cloud. This is hypothetical sensor data coming from a washing machine.

Let's deactivate the debug node and add one on the cloud side.

COPY FOR THE FLOW 1 TO THE FLOW 2 (MSG.PAYLOAD)

Click here to start:



Now we can see the very same data, but now we are officially on the cloud and not pretending we are on an IoT device anymore. We are currently streaming those data into Cloudata at the speed of more than one record per second.

SO NOW LET'S HAVE A BRIEF LOOK AT THE TEST DATA GENERATOR (FLOW1):

The screenshot displays the Node-RED web interface. At the top, a status bar indicates 'Lost connection to server, reconnecting...'. The main workspace shows 'Flow 1' selected, which contains three input nodes: 'Fluid Simulator', 'Voltage Sensor Simulator', and 'Mechanical Sensor Simulator'. Each input node is connected to a corresponding function node ('Fluid data', 'Voltage data', and 'Drum data'). These function nodes are then connected to a central 'Washer01' node, which is marked as 'connected'. A red box highlights the 'Flow 1' tab, and a red arrow points from the 'Washer01' node to the configuration panel on the right.

The configuration panel on the right is titled 'Edit ibmiot out node'. It contains the following settings:

- Authentication: Bluemix Service
- Output Type: Device Event
- Device Type: WashingMachine
- Device Id: Washer01
- Event Type: status
- Format: json
- Data: msg.payload
- QoS: 0
- Name: Washer01

Below the configuration fields, there is a note: 'Note: If there is a property in the message that corresponds to any of the values entered above, then the property in the message takes precedence. See the Info tab for more details. Example JSON device event: {"d":{"myName":"Arduino Uno", "temperature":989}}'. Another note states: 'Note: If there is a property in the message that corresponds to any of the values entered above, then the property in the message takes precedence. See the Info tab for more details. Outdated flows can only report JSON events.'

LET'S HAVE A LOOK AT THE CLOUDANT USER INTERFACE TO SEE WHETHER THE DATA WE ARE STORING ACTUALLY ARRIVES IN THE DATABASE.

- So we open the Bluemix user interface and
- scroll down to the Cloudant service which is bound to our Node-RED application.
- You can easily make autocorrect one since by default it has the same name as the URL of your application.

The image shows a screenshot of the IBM Bluemix console. On the left, a table lists various services. The row for 'noderedkie-cloudantNoSQLDB' is highlighted with a red rectangle. An arrow points from this row to the right, where the Cloudant user interface is shown. The interface displays a list of documents under the 'washing' collection. The documents contain JSON data with fields like 'id', '_id', '_rev', 'value', and 'key'.

Service Name	Service Type	Plan
Blockchain-hj	Blockchain	ibm-blockchain-plan-5-prod
dashDB-iq	dashDB	Entry
discover-iot-try-service	Internet of Things Platform	iotf-service-free
hereGoesYourUniqueAppName-cloudantNoSQLDB	Cloudant NoSQL DB	Lite
hereGoesYourUniqueAppName-iotf-service	Internet of Things Platform	iotf-service-standard
ibm-iotf-py-sim	Internet of Things Platform	iotf-service-free
iottool-cloudantNoSQLDB-devx	Cloudant NoSQL DB	Lite
iottool-iot-devx	Internet of Things Platform	iotf-service-free
noderedkie-cloudantNoSQLDB	Cloudant NoSQL DB	Shared
Object Storage-bt	Object Storage	Free
Personality Insights-kk	Personality Insights	tiered
Predictive Analytics-53	IBM Watson Machine Learning	Free

Click on launch

and washing

JSON Table Include Docs Document ID Options API

washing

All Documents Query Permissions Changes Design Documents

id "1688733bccc1ca8ed25440edcc023fa1"

"_id" "1688733bccc1ca8ed25440edcc023fa1"
"_rev" "1-4c835a5485d24b42c2b778a94b5d73b"
"value" "1-4c835a5485d24b42c2b778a94b5d73b"
"key" "1688733bccc1ca8ed25440edcc023fa1"

id "1688733bccc1ca8ed25440edcc0724b1"

"_id" "1688733bccc1ca8ed25440edcc0724b1"
"_rev" "1-b48a16e63185c123595f0bb1bbbc0c10"
"value" "1-b48a16e63185c123595f0bb1bbbc0c10"
"key" "1688733bccc1ca8ed25440edcc0724b1"

id "1688733bccc1ca8ed25440edcc087048"

ACCESS CLOUDANT DATA ON APACHE SPARK

- In order to access Cloudant data on Apache Spark, we need to obtain the Cloudant credentials.
- So again, we enter the IBM Bluemix dashboard, click on our Node-RED application.

The image shows the IBM Bluemix Cloud Foundry Apps dashboard. The top navigation bar includes the user's account information and a search bar. The main content area displays a list of applications under the heading "All Applications (5)". The applications are listed in a table with columns for NAME, ROUTE, MEMORY (MB), INSTANCES, RUNNING, STATE, and ACTIONS.

NAME	ROUTE	MEMORY (MB)	INSTANCES	RUNNING	STATE	ACTIONS
discover-iot-sample-r...	discover-iot-sample-romeokienzler-1412...	64	1	1	Running	
hereGoesYourUnique...	hereGoesYourUniqueAppName.myblue...	512	1	1	Running	
iot-climate-control-sa...	iot-climate-control-sample-romeokienzle...	768	1	1	Running	
noderedkie	noderedkie.mybluemix.net	512	1	1	Running	
pmqsimulator-romeo...	pmqsimulator-romeokienzler-2310.myblu...	128	1	1	Running	

The application "noderedkie" is highlighted with a red circle. To the right, a detailed view of the "noderedkie" application is shown. It includes a sidebar with navigation options like "Getting Started", "Overview", "Runtime", "Connections", "Logs", and "Monitoring". The main content area displays the application's status as "Your app is running" and provides options to "View App", "Connect Existing", and "Connect New". A red circle highlights the "View Credentials" button for the "noderedkie-cloudantN..." connection. A red arrow points to a modal window showing the credentials for the "cloudantNoSQLDB" connection.

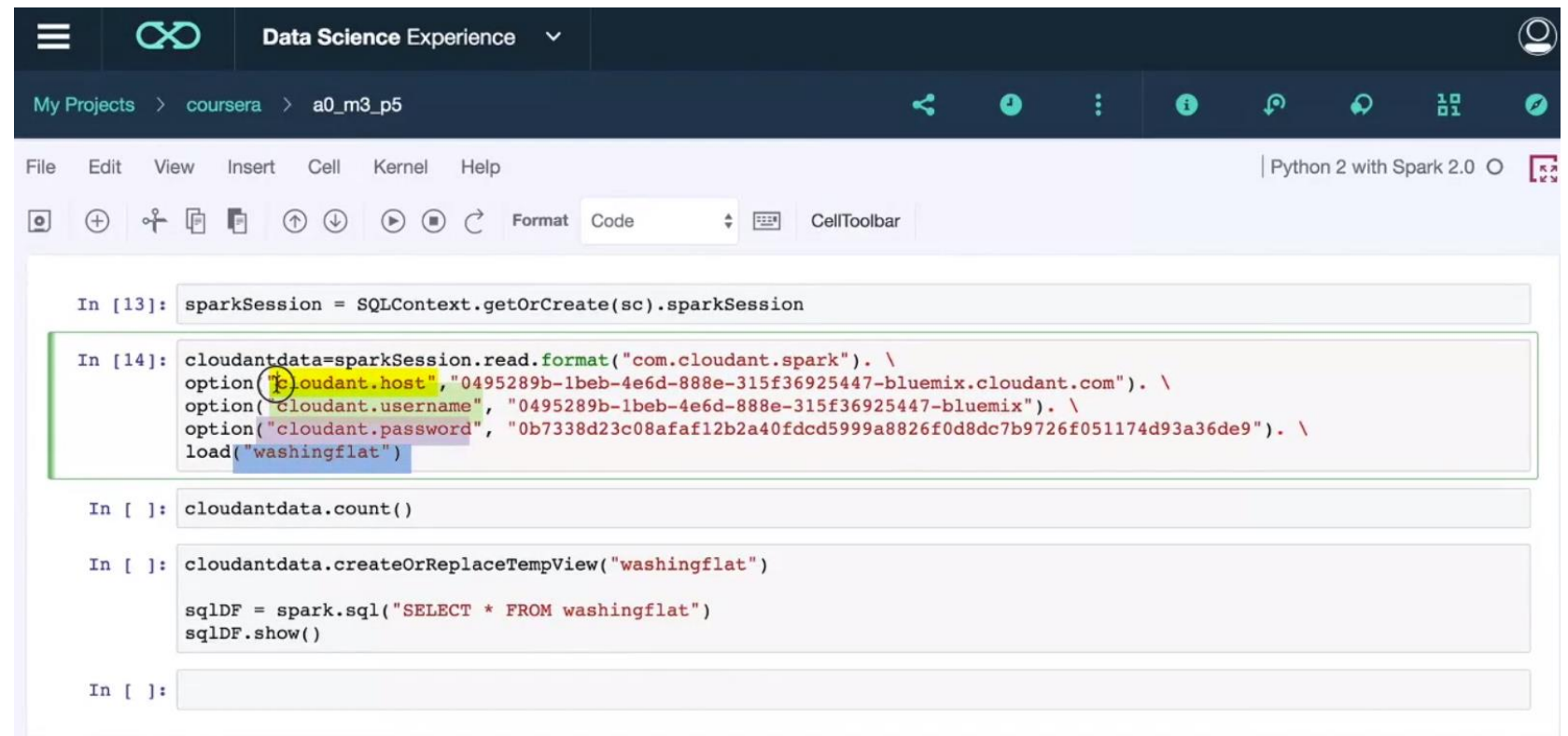
```
"cloudantNoSQLDB": {
  "credentials": {
    "username": "0495289b-1beb-4e6d-888e-31536925447-bluemix",
    "password": "0b7338d23c08faf12b2a40fcd5999a8826f0d8dc7b9726f051174d93a36d69",
    "host": "0495289b-1beb-4e6d-888e-31536925447-bluemix.cloudant.com",
    "port": 443,
    "url": "https://0495289b-1beb-4e6d-888e-31536925447-bluemix:0b7338d23c08faf12b2a40fcd5999a8826f0d8dc7b9726f051174d93a36d69@0495289b-1beb-4e6d-888e-31536925447-bluemix.cloudant.com"
  },
  "syslog_drain_url": null,
  "label": "cloudantNoSQLDB",
  "provider": null,
  "plan": "Shared",
  "name": "noderedkie-cloudantNoSQLDB",
  "runtime": {}
}
```

SO NOW IF WE HAVE ALL INFORMATION TO ACCESS THE DATABASE FROM APACHE SPARK, SO LET'S HAVE A LOOK HOW THIS WORKS.

- There are basically four parameters you have to specify when accessing Cloudant from Apache Spark using the Cloudant connector:

Flow 1" contains a NodeRED application to simulate IoT sensor data. This flow is running in the cloud. In a real scenario, where would this flow normally run?

- On a IoT Gateway and
- On a IoT Device!



The screenshot shows the Databricks Data Science Experience interface. The top navigation bar includes the Databricks logo, the text "Data Science Experience", and a user profile icon. Below this is a breadcrumb trail: "My Projects > coursera > a0_m3_p5". A toolbar with various icons is visible. The main area displays a Jupyter notebook with the following code:

```
In [13]: sparkSession = SQLContext.getOrCreate(sc).sparkSession

In [14]: cloudantdata=sparkSession.read.format("com.cloudant.spark"). \
option("cloudant.host", "0495289b-1beb-4e6d-888e-315f36925447-bluemix.cloudant.com"). \
option("cloudant.username", "0495289b-1beb-4e6d-888e-315f36925447-bluemix"). \
option("cloudant.password", "0b7338d23c08afaf12b2a40fdcd5999a8826f0d8dc7b9726f051174d93a36de9"). \
load("washingflat")

In [ ]: cloudantdata.count()

In [ ]: cloudantdata.createOrReplaceTempView("washingflat")

sqlDF = spark.sql("SELECT * FROM washingflat")
sqlDF.show()

In [ ]:
```

SUMMARY

- IoT data comes from IoT devices
- NodeRED can run on IoT devices
- Easy to simulate test data and run your solution – “the how”
- Straightforward to stream data to Cloudant
- ApacheSpark cloudant connector supports dataframe from Apache Spark