Conserve water with the Internet of Things, Part 3

Customize your Cloudant sensor data for client queries, and build an HTML5 app to display the data in a graph

Bram Havers  
Kai Robbin Weller  
Robert-Jan Sips  
Bill Hymas  

July 18, 2017  
(First published December 16, 2014)

This third tutorial in a three-part series shows how to build a low-cost plant-monitoring solution based on Raspberry Pi. In this final installment, you'll build an HTML5 application that provides a visualization of the sensor data for desktop and mobile devices. You'll create an open data API to make the data available from the Cloudant data store.

Important: IBM Internet of Things Foundation (IoT Foundation) is now named IBM Watson IoT Platform. The IBM Cloud service names have also changed. This article was written using the previous names. The content and images have not been updated. It is provided "as is." Given the rapid evolution of technology, some steps and illustrations may have changed as well.

This third of a three-part tutorial series takes you through construction and configuration of a water-conservation solution — a houseplant soil-moisture monitor — that runs on the Internet of Things. In Part 1, you set up the hardware and used the Internet of Things Foundation Starter on IBM Cloud™ to build a Node-RED app that stores incoming sensor data in a Cloudant database and sends it from there to Twitter. In Part 2, you secured your Node-RED app and set up automated deployment to IBM Cloud.

In this final installment, you'll create an open data API to make the data available from the Cloudant data store. Then you'll use the Data-Driven Documents (D3.js) JavaScript library to build an HTML5 app that provides a visualization of the sensor data for desktop and mobile devices:
“You'll make the data available for the client app to create a graph that shows the last x days' worth of readings for temperature, moisture, and pressure values for a particular sensor.”

**Update: DevOps Services**

The steps in this tutorial series use DevOps Services, which has been replaced by IBM Cloud Continuous Delivery, so these steps are obsolete. To get similar functionality, in IBM Cloud, on your app dashboard overview, click **Enable** in the Continuous Delivery box. To learn more, see the [Add a toolchain to your app](#) tutorial.

**What you'll need to build your solution**

- A [IBM Cloud](https://ibm.com/cloud) account linked to your IBM ID.
- Completion of the steps in [Part 1](#) and [Part 2](#) of this tutorial series.
- HTML and JavaScript experience

**Get the code**

**Step 1. Prepare the Cloudant data store**

In this step, you'll make the sensor readings in the Poseidon Cloudant database available in a format that can be consumed by a client. More specifically, you'll make the data available for the client app to create a graph that shows the last x days' worth of readings for temperature, moisture, and pressure values for a particular sensor.

Every sensor reading that's stored in the database now has the following format:

```
{"_id":"f15b90c35ac432634b7d08d628f284f9","_rev":"2-9d58792b9ad4e39116572ac811a2d569","temperature":21.74,"timestamp":"2014-09-11T20:49:03.950070","altitude":167703.6,"clientID":"ExamplePlantSensor","pressure":1021.52,"longitude":4.8,"moisture":699,"latitude":50.2}
```

By default, all readings are sorted by their ID (`_id`). The first thing to do is to create a new view in Cloudant that sorts the data by using a combined key of sensor name and timestamp. Then you'll be able to do a range lookup for a specific sensor and time range.

1. Look up the host name, user name, and password of your Cloudant instance:
   a. Log in to [IBM Cloud](https://ibm.com/cloud).
b. In the dashboard, click the icon of the app that you created in Part 1 of this tutorial series.
c. Select SDK for Node.js (on the left side).
d. You can see the environment variables for the services in your IBM Cloud instance. (They might take a moment to load.) These variables contain the Cloudant host name, user ID, and password that you need in this tutorial:

```json
{
    "name": "MyPoseidon3:cloudantNoSQLDB",
    "label": "cloudantNoSQLDB",
    "plan": "Shared",
    "credentials": {
        "username": "f8fe0075-9b65-49c5-bfffd-b67c1fe16260-bluemix",
        "password": "1111111111111111111111111b456b35bd2e609740ae54f83662",
        "host": "f8fe0075-9b65-49c5-bfffd-b67c1fe16260-bluemix.cloundant.com",
        "port": 443,
        "url": "https://f8fe0075-9b65-49c5-bfffd-b67c1fe16260-bluemix:
B01e32c10276a103abb374284d34a9bdfad91b456b35bd2e609740ae54f83662@f8fe0075-9b65-
49c5-bfffd-b67c1fe16260-bluemix.cloundant.com"
    }
}
```

Save this code in a file that you can access easily for copy/paste purposes.

2. Create a history view in Cloudant:
   a. In your IBM Cloud instance dashboard, select your Cloudant service.
   b. Click Launch to load the admin user interface for your Cloudant instance.
   c. Click poseidonsensors.
   d. Click the + next to All Design Docs and select New View:

   ![Cloudant Admin User Interface](image)

   3. Fill out the New View form:

   ```
   Views in Cloudant use the MapReduce concept to build the view index. For more information about building your own customized views, see [Guide to Views](#) in the CouchDB documentation.
   ```

   - For Save to Design Document, select New document.
   - In the _/design field, enter sensors.
   - For Index name, enter history.
   - Select None in the optional Reduce field.
   - In the Map function field, enter:
function(doc) {
    if(doc.clientID && doc.timestamp) {
        sensor_values = [0, 0, 0];
        if (doc.temperature) {
            sensor_values[0] = doc.temperature;
        }
        if (doc.moisture) {
            sensor_values[1] = doc.moisture;
        }
        if (doc.pressure) {
            sensor_values[2] = doc.pressure;
        }
        emit([doc.clientID, doc.timestamp], sensor_values);
    }
}

4. Click **Save & Build index**:

5. Test your new view by browsing to:
   https://host/poseidonsensors/_design/sensors/_view/history
   **For example:** https://f8fe0075-9b65-49c5-bffd-b67c1fe16260-bluemix.cloudant.com/poseidonsensors/_design/sensors/_view/history).
   You must provide the username and password from substep 1d. (In Step 2, you'll make this data available as open data without username and password.)
   This request returns all documents:
6. You can limit the list by providing a start and end time through the startkey and endkey parameters. The URL structure becomes:

https://host/poseidonsensors/_design/sensors/_view/history?startkey=[sensor_id, datetime]&endkey=[sensor_id, datetime]

For example: https://f8fe0075-9b65-49c5-bffd-b67c1fe16260-bluemix.cloudant.com/poseidonsensors/_design/sensors/_view/history?startkey=["BramPlantSensor", "2014-10-10T00:00:00.000000"]&endkey=["BramPlantSensor", "2014-11-10T00:00:00.000000"]

7. Log out of Cloudant, and log back in to refresh the Cloudant admin UI. Now you can see your new history view in the Views list:

8. The history view returns all the data you need to create a graph, but you can make it more compact by transforming it into comma-separated value (CSV) format (by using CouchDB list functions):

   a. Click All Design Docs in the Cloudant admin interface.
   b. Click Edit (pencil icon in right upper corner of the document) to edit the design document that you created in this step.
   c. Add the following code to the design document, immediately following the closing bracket of the views code section (don't forget the comma right after the closing bracket). See the GitHub repository for the full example code.

```javascript
"lists": {
"csv": "function(head, req){ start({ 'headers': { 'Content-Type': 'text/csv' } });
send('timestamp,temperature,moisture,pressure\u000A'); while(row = getRow()){ send(row.key[1] + ',' + row.value[0] + ',' + row.value[1] + ',' + row.value[2] + '\u000A'); } }",
"json": "function (head, req) { provides('json', function() { var results = []; while (row = getRow()) { results.push({ name: row.key }); }send(JSON.stringify(results)); }); }"
}
```
9. As an alternative (no need to do this now), you can resubmit the complete design document (including history and lists views) without using the admin interface. Open Tutorial3/Cloudant/creating_views.json in edit mode in your cloned PoseidonProject repository. Change the revision number (_rev) to be similar to the current design document in Cloudant (so that Cloudant knows that you're updating the previous version). Upload the file: curl -X PUT https://host/poseidonsensors/_design/sensors --data-binary @sensors_design.json. Cloudant will respond with the result and revision number of the design document created.

Test the new lists view in your browser with the URL structure: https://host/poseidonsensors/_design/sensors/_list/csv/history?startkey=[sensor_id, datetime]&endkey=[sensor_id, datetime]
For example: https://f8fe0075-9b65-49c5-bffd-b67c1fe16260-bluemix.cloudant.com/poseidonsensors/_design/sensors/_list/csv/history?startkey=["BramPlantSensor", "2014-10-10T00:00:00.000000"]&endkey=["BramPlantSensor", "2014-11-10T00:00:00.000000"]
The URL returns a CSV file with sensor readings:

![CSV file with sensor readings](https://via.placeholder.com/150)

10. Create a Cloudant view to expose a JSON list of sensors.
   a. In the Cloudant DB, click the + next to _design/sensors and select New View.
   b. For Index name, enter summary.
   c. In the Map function field, enter:
      
      ```javascript
      function(doc) {
        if(doc.clientID) {
          emit(doc.clientID, null);
        }
      }
      ```
   
   d. Set the Reduce function to _count and save.
   e. Verify the view works. For example: https://febceb81-4238-4826-a773-5284a4677024-bluemix.cloudant.com/poseidonsensors/_design/sensors/_view/summary?group=true
   f. In the Cloudant web client, go to All design docs and edit the _design/sensors document.
   g. In the lists section, add the json list after the "csv" list definition (don't forget to add a comma at the end of the previous line) and save:
      
      ```javascript
      "json": "function (head, req) { provides('json', function() { var results = []; while (row = getRow()) { results.push({ name: row.key });}send(JSON.stringify(results));});}"
      ```
   h. Verify that the list works. For example: https://febceb81-4238-4826-a773-5284a4677024-bluemix.cloudant.com/poseidonsensors/_design/sensors/_list/json/summary?group=true
   i. Verify that the days parameter works by just adding it to the URL. For example: http://poseidontutorial-tftx.mybluemix.net/sensors/F5TX/history?days=1

That's it for this step. You have finished preparing your data store.
Step 2. Create the API

Now you'll design two APIs that enable the visualization app to retrieve its data. The first API retrieves a list of available sensors, and the second retrieves the data from a specific sensor. This table provides the specifications for these APIs:

<table>
<thead>
<tr>
<th>URL</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/sensors</td>
<td>No parameters</td>
<td>Returns a JSON list of all sensors. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ &quot;name&quot; : &quot;BramPlantSensor&quot; },</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ &quot;name&quot; : &quot;KaiPlantSensor&quot; },</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ &quot;name&quot; : &quot;KaisGrove&quot; },</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ &quot;name&quot; : &quot;TheGruPi&quot; },</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ &quot;name&quot; : &quot;TheGruSensor&quot; }</td>
</tr>
<tr>
<td>/sensors/{sensor}/history</td>
<td></td>
<td>Returns a CSV list containing sensor readings for the specified {sensor}. The result is limited to the last 14 days by default, but it can be overwritten. The history API returns a CSV list containing a history of sensor readings with the timestamp, temperature, moisture, and pressure levels. For example:</td>
</tr>
<tr>
<td>sensor</td>
<td>Name (unique ID) of the sensor</td>
<td></td>
</tr>
<tr>
<td>days</td>
<td>Optional. Number of days of sensor readings to return</td>
<td></td>
</tr>
</tbody>
</table>

Use Node-RED to implement the APIs:

1. Open the PoseidonProject DevOps Services repository and copy the contents of the / Tutorial3/Node-red/API_cloudant_proxy_flow.json file to the clipboard.
2. Browse to your IBM Cloud application's public page and click Go to your Node-RED flow editor.
3. Click the + to create a new worksheet, double-click the tab, and rename the worksheet to REST API.
4. Select Import > Clipboard from the menu, paste the code, click Ok, and place the proxy flow somewhere on your worksheet.
5. Click each of the "HTTP in" nodes (Sensor History and Sensor List) to see its configuration in the info tab.
6. Edit each of the two Convert To Cloudant URL function nodes (double-click the node) and change the cloudantHost variable to match your Cloudant instance.
7. Update the Cloudant GET node to use basic authentication:
   - Leave the URL field empty (the URL is provided by the previous node).
   - Select the Use basic authentication? check box.
• For **Username**, enter the user name of your Cloudant instance.
• For **Password**, enter the password of your Cloudant instance.

8. The Add CORS Header node adds the Access-Control-Allow-Origin HTTP header to the HTTP response of the REST call. This header enables Cross Origin Resource Sharing so that the API can be called from web pages in a different domain from the one hosting the API.

Explore the properties of the Add CORS Header function node and the HTTP Response node.

9. Click **Deploy** to make this proxy flow active.


Your open data API is now ready for use without authentication.

**Step 3. Create the HTML5 app with D3.js**

1. Export the HTML5 code to your local PC:
   a. Browse to the PoseidonProject DevOps Services repository.
   b. Click **EDIT CODE**. (Log in if necessary.)
   c. Browse to Tutorial3/node-app/public, right-click the public folder in the project tree, and select **Export > Zip**.

2. Import the code into your running IBM Cloud environment:
   a. Browse to the DevOps Services repository for your running IBM Cloud instance. If you forgot the name, go to your IBM Cloud dashboard and click **EDIT CODE**:

   ![PoseidonServer Dashboard](image)

   b. Browse to //public. Right-click public and select **Import File or Zip archive**. (By default the import action does not overwrite files, so you'll receive an error message. You only want to add a file, so you can ignore the message.)

3. Commit the files to Git to trigger the autodeployment process:
   a. In the code editor for your DevOps Services project, click the Git icon (second from the top on the left).
   b. Select all changes.
   c. Enter a commit message (such as *new html5 app*) and click **COMMIT**.
   d. Click **SYNC** to push and sync the commits. This action triggers autodeploy.
e. Check deployment status by clicking **BUILD & DEPLOY** on your project's overview page.

4. Test the app by browsing to `http://poseidon_host/sensors.html` (for example, `http://poseidon.mybluemix.net/sensors.html`). You'll see a list of sensors that have already sent sensor readings:

5. Click one of the sensors to view the graph created with D3.js.

**Conclusion to this series**

The water-monitoring solution originated with the Poseidon project, a voluntary initiative that aims to reduce global water usage.

In this final tutorial, you learned how to create customized views on Cloudant and how to create an API proxy using Node-RED. You built an app that consumes the API and presents a graph of the sensor data.

We hope that you have enjoyed this series of Poseidon tutorials. We challenge you to improve the code and contribute your changes back. For example, you could send notifications when your plant needs watering.
Related topics

- Creating apps with the Internet of Things Starter
- How IBM’s Node-RED is hacking together the internet of things
- Data visualization with D3
- See IBM Cloud in action
- The Internet of Things service
- The Cloudant NoSQL service
- The SDK for Node.js runtime
- Internet of Things
- Cloudant
- Node-RED
- JSON

© Copyright IBM Corporation 2014, 2017
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)