

Практикум по быстрому прототипированию решений Интернета вещей

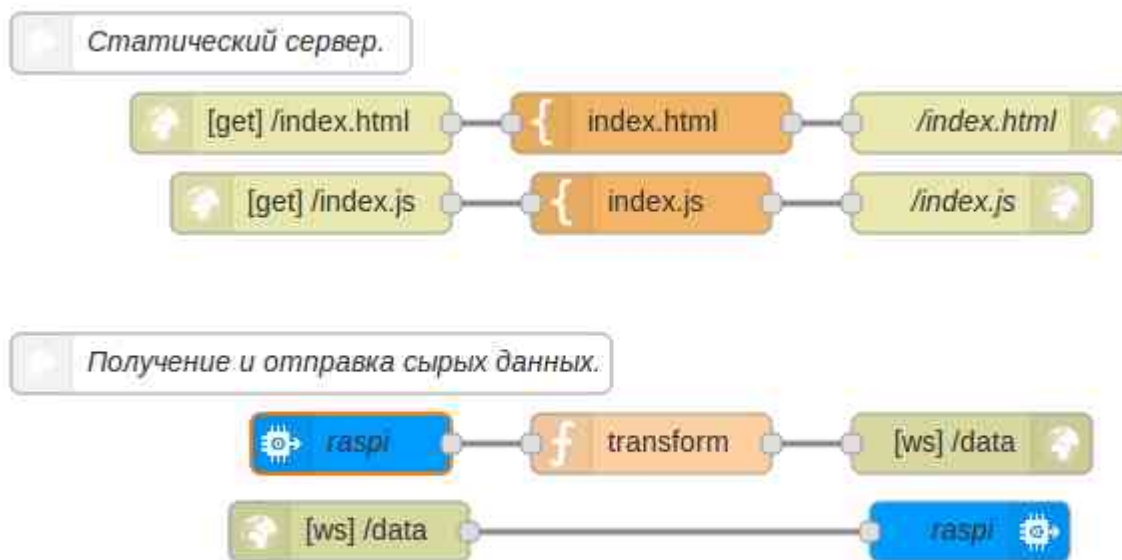
Визуализация

Сервер

Из рабочего пространства проекта перейдите в приложение SDK for NODE.js. В левом верхнем углу нажмите на ссылку для перехода в SDK NodeRED.

Дальнейшие действия поясняют процесс создания приложений в данном SDK.

Экспортируйте в NodeRED следующий код: [Пример NodeRED](#)



Измените настройки блока `raspi` следующим образом:

Authentication: Bluemix Service

Device ID: mac адрес вашего устройства.

Точкой входа в веб-приложение является файл `index.html`, для получения которого браузер выполняет http-запрос к нашему серверу. Поэтому нам необходимо воспользоваться node-red узлами `input/http` и `output/http`:

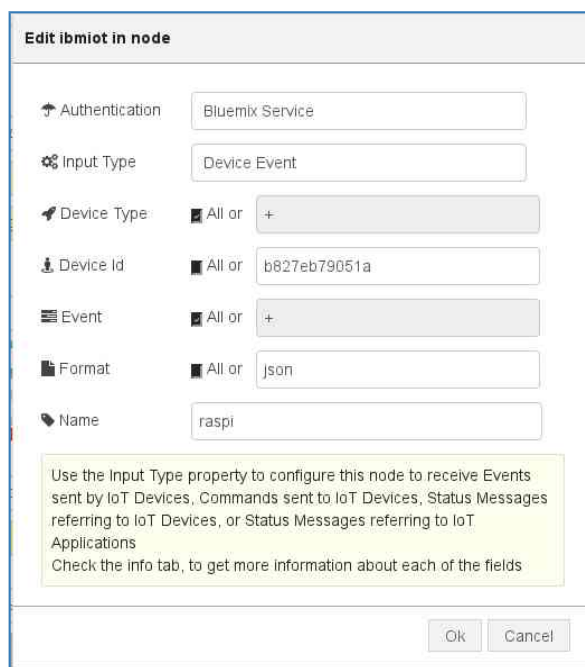


Статический сервер.

Теперь вход в приложение может быть осуществлён по адресу `http://YOU_PROJECT_NAME.eu-gb.mybluemix.net/index.html` (например, <http://example-hack.eu-gb.mybluemix.net/index.html>).

Содержимое `index.html` и `index.js` подробно рассматривается в следующем разделе. Полный код: [index.html](#) и [index.js](#). Вы можете скопировать их в ваш проект, выбрав в верхнем правом углу NoDeRED редактор.

Теперь нам необходимо настроить взаимодействие с raspberry, т.е. получение и отправку данных. Для этого используется node-red узлы `input/ibmiot` и `output/ibmiot`:



Настройки узла `input/ibmiot`.

Edit ibmiot out node

Authentication:

Output Type:

Device Type:

Device Id:

Command Type:

Format:

Data:

Name:

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}}

Настройки узла output/ibmiot.

Теперь мы можем взаимодействовать с raspberry:



Получение и отправка данных.

Узел трансформации transform просто преобразует данные для пересылки клиенту:

```

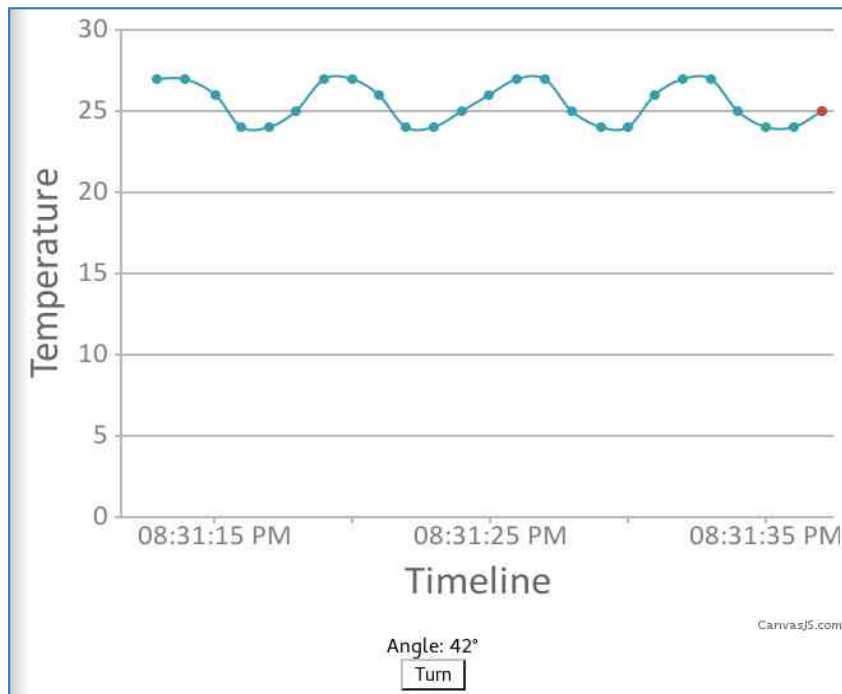
msg.payload = {
  topic: msg.eventType,
  data: msg.payload
};

return msg;

```

Клиент

В данном разделе рассмотрим вопрос реализацию визуализации на клиенте.



Визуализация потоковых данных.

В ответ на запрос `index.html`, сервер отдаёт страницу, в которой содержатся:

1. Ссылка на библиотеку для рисования графиков:

```
<script
src="//cdnjs.cloudflare.com/ajax/libs/canvasjs/1.7.0/canvasjs.min.js"></script>
```

2. Ссылка на наш код, получающий данные:

```
<script src="index.js" defer></script>
```

3. Некоторые элементы, используемые в скрипте:

```
<div id="graph"></div>
<div>Angle: <span id="angle">??</span>°</div>
<input type="button" id="turn-btn" value="Turn on">
```

Рассмотрим подробнее логику клиента.

Для начала необходимо получить данные с `bluemix`. Для этого мы открываем сокет:

```
var socket = new WebSocket('ws://' + window.location.host + '/data');
```

Теперь нам потребуется обработчик поступающих данных:

```
socket.onmessage = function(e) {
  var item = JSON.parse(e.data);

  switch (item.topic) {
    case 'temperature':
      processTemp(item.data);
      break;
    case 'angle':
      processAngle(item.data);
      break;
  }
};
```

График температуры

После получения актуальной температуры необходимо отобразить новое значение на графике:

```
var $graph = document.querySelector('#graph');

// Массив, содержащий все точки.
var actual = [{x: new Date}];

var chart = new CanvasJS.Chart($graph, {
  axisX: {title: "Timeline"},
  axisY: {title: "Temperature"},
  data: [{
    type: "spline",
    dataPoints: actual
  }]
});

chart.render();

function processTemp(temp) {
  var now = new Date;

  // Будем хранить только последние две минуты.
  if (+now - actual[0].x > 2 * 60 * 1000)
    actual.shift();

  var point = {x: now, y: temp};
  actual.push(point);
  chart.render();
}
```

Угол поворота

После получения актуального значения угла достаточно просто обновить элемент:

```
var $angle = document.querySelector('#angle');

function processAngle(angle) {
  $angle.innerHTML = angle;
}
```

Кнопка

При нажатии на кнопку нам важен сам факт нажатия, поэтому достаточно посылать `true`:

```
var $button = document.querySelector('#turn-btn');
$button.onclick = function() {
  socket.send(true);
};
```

Аналитическая обработка данных

В этой части проекта необходимо выполнить статистическую обработку получаемых данных. Для этого потребуются следующие компоненты инфраструктуры:

- В платформе Bluemix реализовать сервис хранения данных в БД dashDB.
- Разработать структуру таблиц базы данных и SQL скрипты для добавления новых данных и удаления устаревших данных.
- Разработать поток Node-Red, реализующий запуск SQL скриптов.
- Выполнить проверку работоспособности потока с использованием консоли административной консоли dashDB.
- Выполнить разработку аналитического скрипта на языке R в среде RStudio.
- Разработать потока обработки Node-Red для запуска R скрипта.

Дополнительная информация:

[Язык программирования R](#)

[Краткая справка по командам языка R](#)

[Описание IDE RStudio](#)

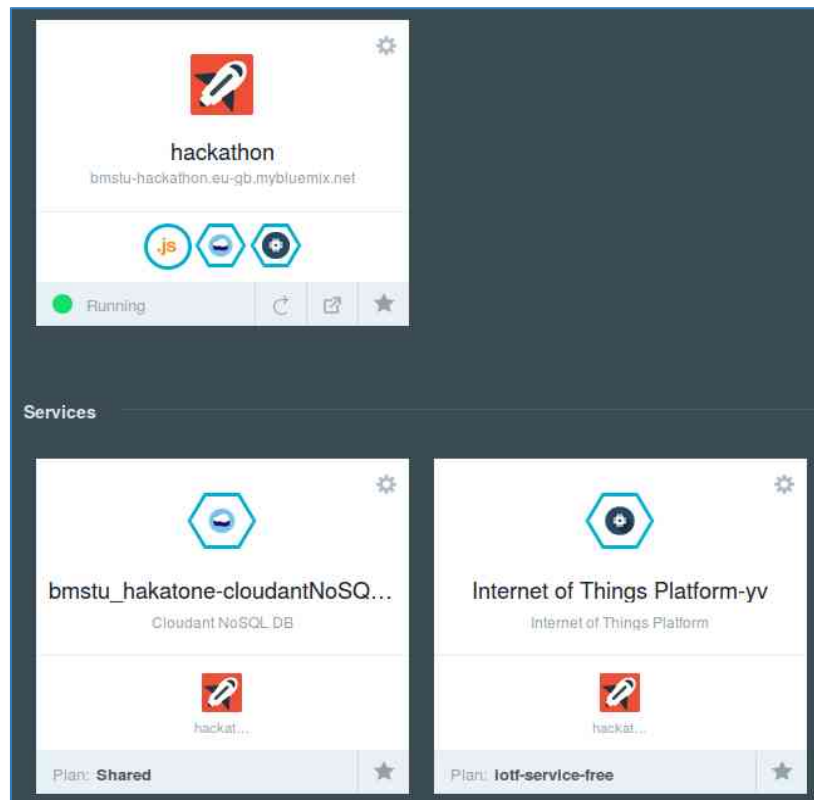
[Инструкции определения данных на языке DDL](#)

[Краткий справочник по командам SQL](#)

Работа с сервисом dashDB

К началу этапа рабочая область проекта представляет собой три взаимосвязанных компонента:

- Сервис IoT Foundation для реализации функций брокера MQTT.
- Сервис CloudantDB для хранения настроек IoT Foundation и Node-Red.
- Приложения JavaScript в Node-Red сервере приложений.



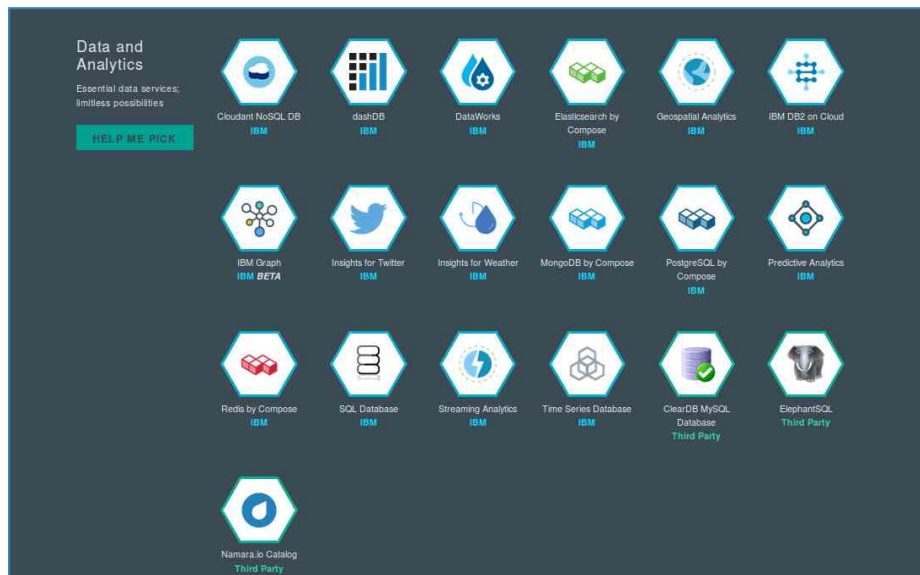
Рабочая область проекта.

Сервис dashDB представляет собой интегрированные компоненты для реализации функций хранения и аналитической обработки данных с помощью языка R. Сервис позволяет создавать и контролировать состояние SQL базы данных dashDB, содержит набор готовых скриптов на языке R, позволяет создавать и отлаживать пользовательские скрипты.

Добавление сервиса в проект

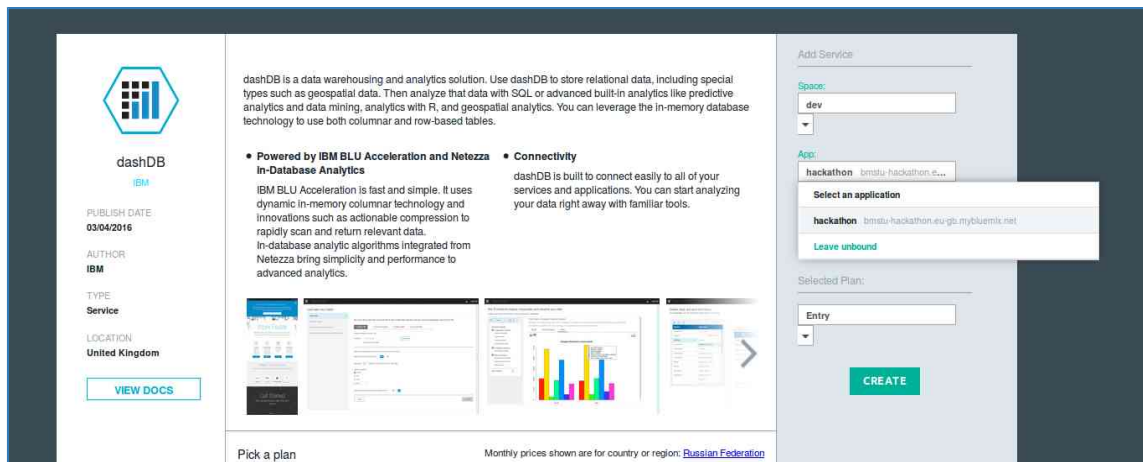
Выполним добавление сервиса dashDB.

На вкладке Catalog в секции Data and Analytics необходимо выбрать сервис dashDB.



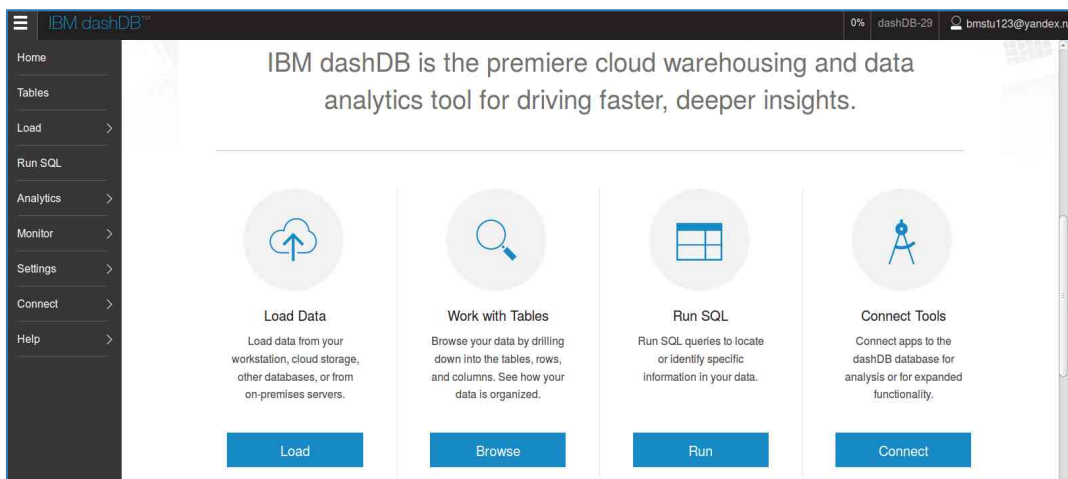
Добавление сервиса dashDB.

В следующем окне вводятся поля Dev, App, Service. Все поля кроме App можно оставить без изменений. В поле App в выпадающем списке выберете имя вашего приложения Node-Red. Связывание позволит выполнять обращения к dashDB со стороны приложений Node-Red (можно выполнить связывание позднее).



Связывание dashDB с приложением Node-Red.

Нажмите кнопку Create. После создания сервиса появится возможность перейти в консоль управления. Для этого нажмите на кнопку Launch.

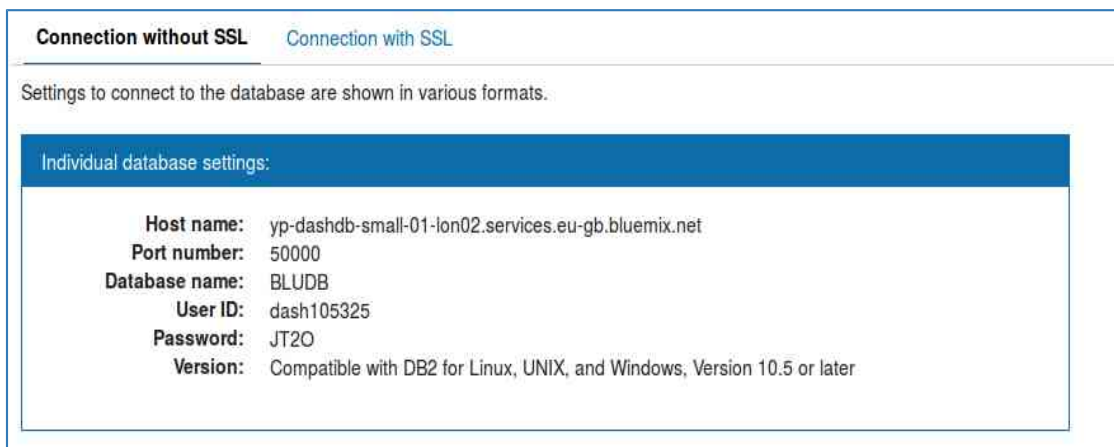


Консоль управления dashDB.

Для работы с сервисом необходимо определить параметры:

- *User ID*
- *Host name*
- *Password*

Указанная информация доступна на вкладке Connect в пункте Connect Information. Указанная информация понадобится впоследствии.



Параметры для подключения к сервису dashDB.

Создание таблицы

На вкладке Tables объединены функции управления структурой базы данных. При создании сервиса автоматически создается новая база с именем, совпадающим с полем User ID (в примере: DASH105325).

Добавим в базу таблицу TEMP для хранения данных.

Для этого необходимо выбрать пункт Add Table и в открывшемся окне ввести код DDL.

```
CREATE TABLE TEMP (  
    TIME BIGINT NOT NULL PRIMARY KEY,  
    TEMP DOUBLE NOT NULL
```

);
Поле TIME предусмотрено для хранения метки времени.

Запуск потоковой записи первичных данных

В приложении Node-Red необходимо связать блок IoT Foundation с функциональными блоками, формирующими структуру объекта payload, после чего передать его в блок output/dashdb. В блоке dashDB указать поле Service dashDB-xx (название сервиса dashDB).



Поток обработки для записи данных в dashDB.

Обратите внимание, что в функциональных блоках название ключей в структуре payload должно совпадать с полями таблицы базы данных.

Например:

```
msg.payload = {  
  TIME: Date.now(),  
  TEMP: msg.payload  
};
```

```
return msg;
```

Проверим работоспособность приложения (кнопка Deploy). В консоли управления dashDB на вкладке Tables выберете таблицу TEMP и Browse Data.

Данные от сенсоров должны быть выбаны на экран.

TIME	TEMP
1458577871154	24
1458577900207	25
1458577929257	25
1458577958316	23
1458577982996	23
1458578011811	27

Проверка работоспособности скриптов SQL в консоли dashDB.

Удаление данных из таблиц

Все полученные данные накапливаются в таблице TEMP. Так как время аналитической обработки зависит от объемов данных, выполним удаление устаревших строк из таблиц. Для

этого добавим следующий поток обработки, содержащий SQL скрипты



Удаление устаревших данных из таблиц dashDB.

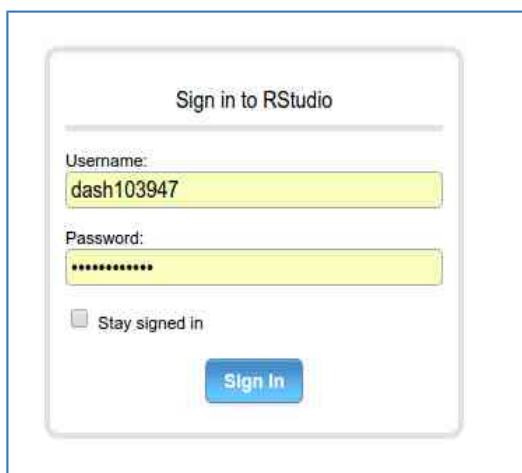
Скрипт для удаления данных старше одного часа:

```
DELETE FROM TEMP WHERE TIME <= (SELECT MAX(TIME) FROM TEMP) - 3600000;
```

Создание скрипта в RStudio

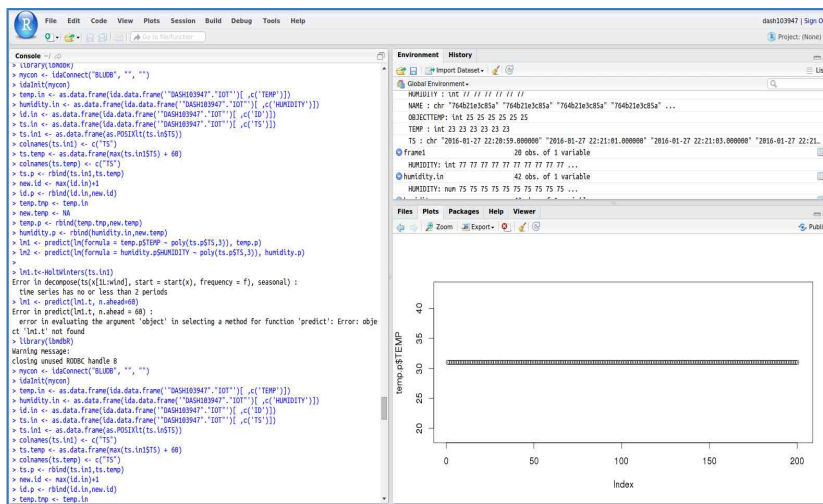
В консоли dashDB перейти в пункты Analytics и далее R Scripts.

Выбрать пункт RStudio.



Ввод User ID и Password для входа в RStudio.

В результате будет открыто окно RStudio, в котором может выполняться пошаговая отладка команд на языке R.



Окно среды RStudio.

Подробнее о работе с IDE RStudio можно узнать [тут](#)

Выполним следующий скрипт (скрипт может быть вставлен в окно Console):

```

library(ibmdbR)
mycon <- idaConnect("BLUDB", "", "")
idaInit(mycon)

# Загрузить данные из таблицы (запись данных в таблицу осуществляется также SQL
запросом)
data <- idaQuery('SELECT TIME, TEMP from TEMP', as.is = FALSE)
fit <- lm(TEMP ~ TIME, data)

# Предсказать на минуту вперед
new <- data.frame(TIME = max(data$TIME) + 60000)
new$TEMP = predict(fit, new)

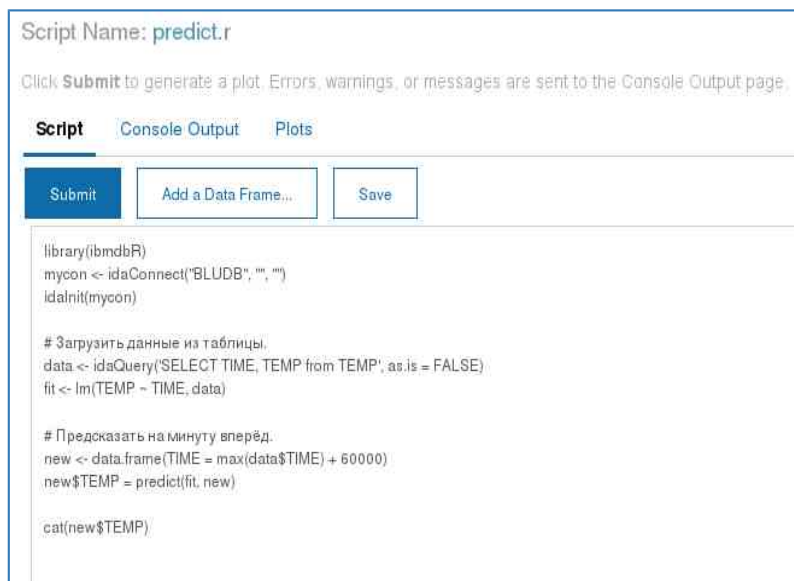
cat(new$TEMP)

```

В результате скрипт будет выполнен, а все использованные фреймы могут быть проанализированы на вкладке Environment.

Запись скрипта в файловую систему окружения dashDB

Для автоматического запуска разработанного скрипта необходимо сохранить его в рабочем пространстве проекта. Для этого необходимо перейти в консоль управления dashDB в пункт Analythics и пункт RScripts. Далее необходимо создать новый скрипт (+) и выполнить вставку кода R. Сохраним скрипт под именем predict.r.



Запись скрипта.

Также можно выполнить тестовый запуск скрипта, нажав на кнопку Submit.

Запуск скрипта по расписанию из node-red

В редакторе NodeRed создайте следующий поток:



Запуск скрипта по расписанию.

URI creator нужен для составления запроса:

```
var source = encodeURIComponent('source("~/predict.r")');
msg.payload = 'cmd=RScriptRunScript&command='
              + source
              + '&fileName=&profileName=BLUDB&userid=dash107216';
msg.headers = {'content-type': 'application/x-www-form-urlencoded'};
return msg;
```

В данном коде необходимо заменить dash107216 на user ID пользователя dashDB (например DASH015794).

The screenshot shows the 'Edit http request node' dialog box. The fields are: Method: POST; URL: https://yp-dashdb-small-01-lon02.services.eu-gb.l...; Use basic authentication?: checked; Username: dash107216; Password: masked with dots; Return: a parsed JSON object; Name: R Script. A tip at the bottom states: 'Tip: If the JSON parse fails the fetched string is returned as-is.' There are 'Ok' and 'Cancel' buttons at the bottom right.

Узел запроса.

В данном узле необходимо выбрать способ аутентификации ([v] Use basic authentication?), в поле URL выставить правильный host name адрес dashDB приложения, в полях Username и Password указать значения user ID и password из настроек dashDB. Интервал запуск задается в начальном узле. Для этого в поле Repeat нужно указать временной интервал между генерацией сообщений (например, 20 секунд).

Визуализация предиктивных данных

Разумно отображать предиктивную прямую на том же графике, что и актуальную информацию о температуре.

Добавим в функцию `socket.onmessage` обработку потока предиктивных данных:

```
switch (item.topic) {
  case 'temperature':
    // ...
  case 'predicted':
    processPredicted(item.data);
    break;
}
```

И добавим поток в инициализацию библиотеки для графиков:

```
var actual = [{x: new Date}];
var predicted = [];

var chart = new CanvasJS.Chart($graph, {
  axisX: {title: "Timeline"},
  axisY: {title: "Temperature"},
  data: [{
    type: "spline",
    dataPoints: actual
  }, {
    type: "line",
    dataPoints: predicted,
    lineThickness: 1,
    lineDashType: 'dashDot'
  }]
});
```

Будем отображать предиктивные данные как прямую от текущей точки к предсказанной. Поэтому `predicted` всегда будет содержать только две точки. Предсказанная точка корректируется каждый раз при поступлении новых предиктивных данных:

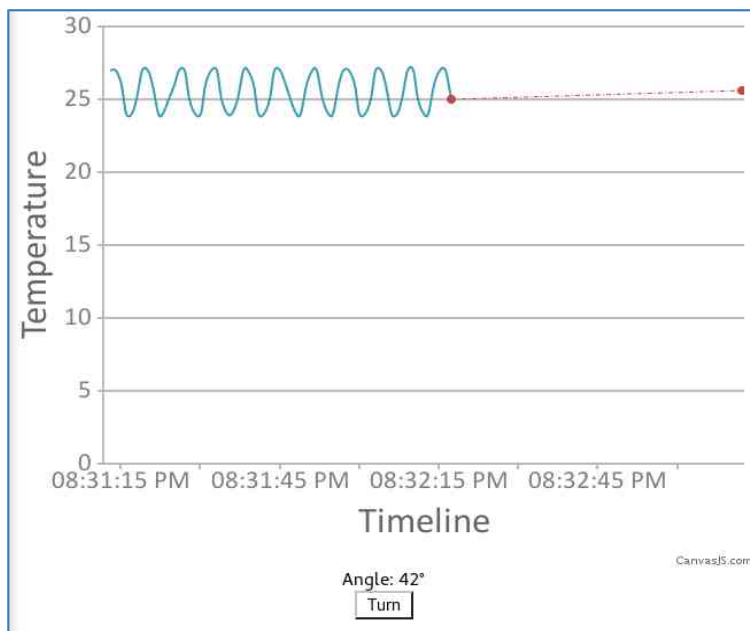
```
function processPredicted(temp) {
  predicted[1] = {
    x: new Date(Date.now() + 60 * 1000),
    y: temp
  };

  chart.render();
}
```

А текущая (`predicted[0]`) при поступлении актуальных:

```
function processTemp(temp) {
  // ...
  actual.push(point);
  predicted[0] = point;

  chart.render();
}
```



Результат визуализации.

Полный код примера проекта вы можете найти тут: [Пример проекта NodeRED](#)