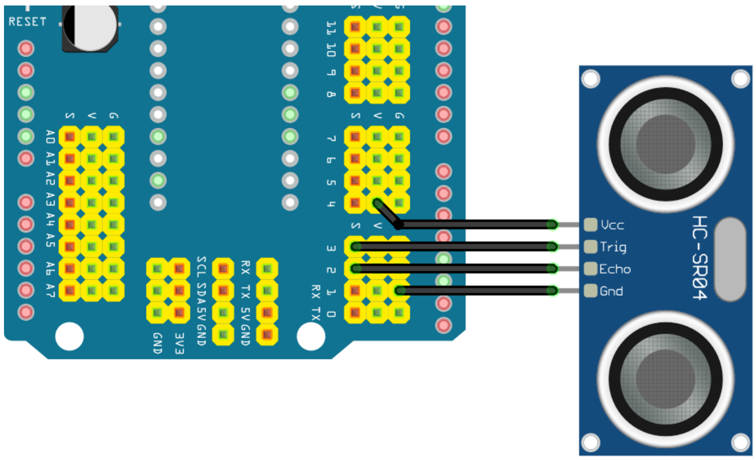
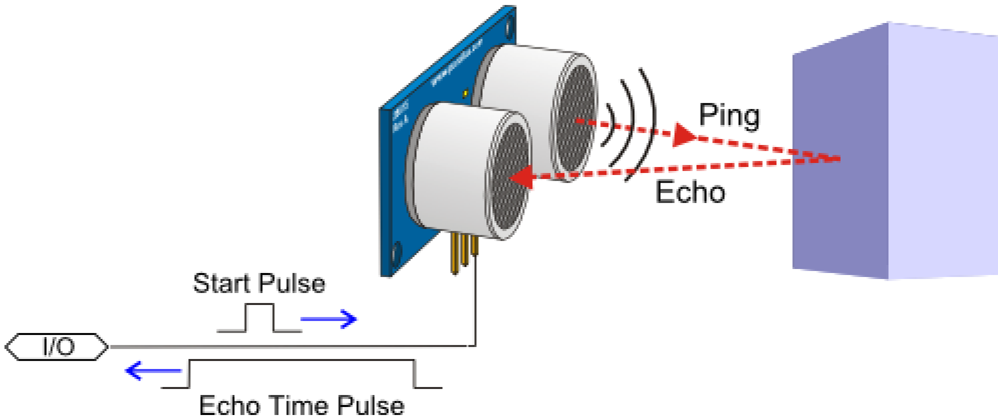
Task 1. Connect the circuit as shown in the picture:



HC-SR04	Arduino
Vcc	V
Trig	3
Echo	2
Gnd	G

Note: the following code is not a complete project allowing You to compile and program the Arduino board. On your own, You should put individual pieces of code in functions `setup()` and `loop()`. Results (`distance` variable) should be sent to the PC.

```
#define TriggerPin 3
#define EchoPin 2

long distance;

pinMode(TriggerPin,OUTPUT);
pinMode(EchoPin,INPUT_PULLUP);

digitalWrite(TriggerPin,LOW);
delayMicroseconds(2);
digitalWrite(TriggerPin,HIGH);
delayMicroseconds(10);
digitalWrite(TriggerPin,LOW);
distance = pulseIn(EchoPin,HIGH)/58.2;
```



Pin	Description
Vcc	Power supply +5V
Trig	Trigger input
Echo	Data output
GND	Power supply - ground

The basic parameters of the HC-SR04 module are as follows:

- power supply: 5[V];
- average current consumption: 15[mA];
- range: from 2[cm] to 200[cm] (in reality up to ca. 3[m]);
- resolution: 0.3[cm];
- measuring angle: 30[°];
- dimensions: 45[mm] x 20[mm] x 15[mm].

What's new:

`delayMicroseconds` function, `pulseIn` function

Task 2. Write a function called *hcsr04_measurement* that returns a result of distance measurement. Use the circuit from **Task 1**. Send results to Your computer.

```
#define BAUDRATE 115200
#define TriggerPin  3
#define EchoPin     2

long response;

void setup() {
 Serial.begin(BAUDRATE);
 pinMode(TriggerPin,OUTPUT);
 pinMode(EchoPin,INPUT_PULLUP); }

void loop() {
 response = hcsr04_measurement(TriggerPin,EchoPin);

 Serial.print("Distance: ");
 Serial.print(response);
 Serial.println("[cm]");
 delay(500); }

long hcsr04_measurement(int T_Pin, int E_Pin) {
 digitalWrite(T_Pin,LOW);
 delayMicroseconds(2);
 digitalWrite(T_Pin,HIGH);
 delayMicroseconds(10);
 digitalWrite(T_Pin,LOW);
 return pulseIn(E_Pin,HIGH)/58.2; }
```

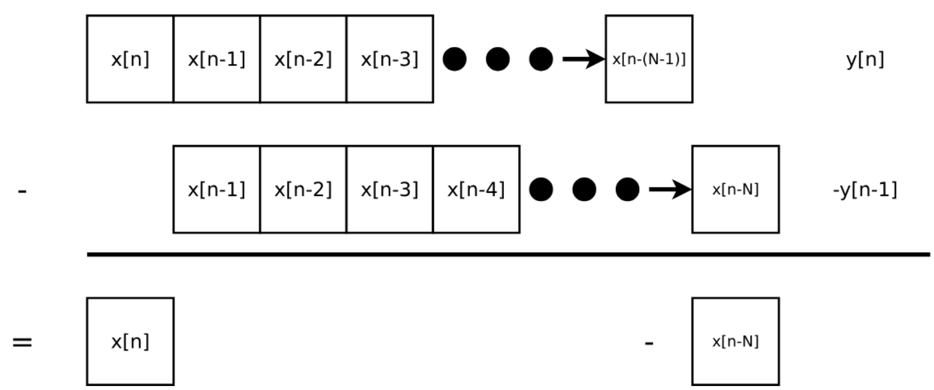
Task 3. Improve results by removing false readings. Calculate an average of a set of 4 distance readings.

```
#define BAUDRATE 115200
#define TriggerPin 3
#define EchoPin 2
#define MAX_RANGE 280

long distance;
void setup() {
 pinMode(TriggerPin,OUTPUT);
 pinMode(EchoPin,INPUT_PULLUP);
 Serial.begin(BAUDRATE); }

void loop() {
  long avg = 0;
  for(int i=0; i<4; ) {
   distance = hcsr04_measurement(TriggerPin,EchoPin);
   if(distance < MAX_RANGE) {
     i++;
     avg+=distance; }
   delay(50); }
   Serial.print("Distance to the target: ");
   Serial.println(avg/4); }

long hcsr04_measurement(int T_Pin,int E_Pin) {
 digitalWrite(T_Pin,LOW);
 delayMicroseconds(2);
 digitalWrite(T_Pin,HIGH);
 delayMicroseconds(10);
 digitalWrite(T_Pin,LOW);
 return pulseIn(E_Pin,HIGH)/58.2; }
```

Task 4. Modify Your code from Task 4 by changing the arithmetic mean to the moving average(MA).

No	Value (x)	Moving average
1	x[1] = 120	X
2	x[2] = 124	X
3	x[3] = 132	X
4	x[4] = 128	y[4] = (120 + 124 + 132 + 128)/4
5	x[5] = 130	y[5] = (124 + 132 + 128 + 130)/4
6	x[6] = 132	y[6] = (132 + 128 + 130 + 132)/4
7	x[7] = 126	y[7] = (128 + 130 + 132 + 126)/4

MA implementation:



Assuming 4 samples (N = 4):

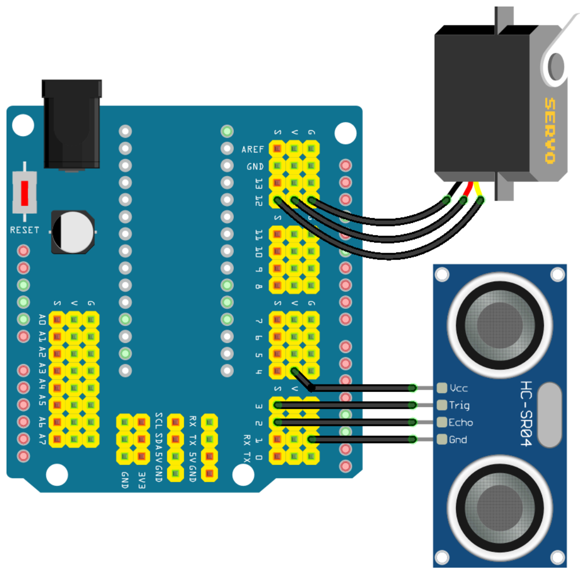
$$y[n] = y[n - 1] + x[n] - x[n-4]$$

The general equation for **N** samples equals:

$$y[n] = y[n - 1] + x[n] - x[n-\mathbf{N}]$$

```
long results[5];

void loop() {
  long m_avg = 0;
  int n=0;
  int i=0;

  for(i=0; i<4; ) {
   distance = hcsr04_measurement(TriggerPin,EchoPin);
   if((distance > 0) && (distance < MAX_RANGE)) {
     results[i]=distance;
     i++;
   }
   delay(100); }

  for(i=0; i<4; i++) m_avg += results[i];
  m_avg/=4;

  while(1) {
   do {
    results[4] = hcsr04_measurement(TriggerPin,EchoPin);
    delay(100);
   } while(results[4] > MAX_RANGE);

    m_avg = m_avg + (results[4] - results[n++])*0.25;
    if(n==4) n=0;
    results[n] = results[4];

    Serial.print("Distance to the target: ");
    Serial.println(m_avg);
 }
}
```

Task 5. Build a prototype of a device that allows controlling servo shaft position with a distance sensor.



Task 6. Build a prototype of a device that uses LEDs to inform about the distance to a target.

Task 7. Introduction to JSON.

JSON (**J**ava**S**cript **O**bject **N**otation) is a language-independent, open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs.

To work with JSON objects, add *`ArduinoJSON`* library to Arduino IDE:

```
Sketch -> Include Library -> Manage Libraries
```

In the example the following JSON object is used:

```
{ "name" : "HC-SR04", "value" : 0.0 }
```

```
#include<ArduinoJson.h>
#define BAUDRATE 115200
```

```
#define TriggerPin 3
#define EchoPin 2
#define MAX_RANGE 280

const int capacity = JSON_OBJECT_SIZE(2);
StaticJsonDocument<capacity> sensor;

void setup() {
 Serial.begin(BAUDRATE);
 sensor["name"] = "HC-SR04";
 sensor["value"] = 0.0;
 pinMode(TriggerPin,OUTPUT);
 pinMode(EchoPin,INPUT_PULLUP);
}

long p_millis = 0;
long distance;
long avg_distance = 0;
#define DELAY 1000

void loop() {
 if(millis() - p_millis > DELAY) {
  p_millis = millis();
  avg_distance = 0;
  for(int i=0; i<4; ) {
   distance = hcsr04_measurement(TriggerPin,EchoPin);
   if(distance < MAX_RANGE) {
     i++;
     avg_distance += distance; }
   delay(100);
  }
 sensor["value"] = avg_distance;
 serializeJson(sensor,Serial);
 Serial.print("\n");
 }
}

long hcsr04_measurement(int T_Pin, int E_Pin) {
 digitalWrite(T_Pin,LOW);
 delayMicroseconds(2);
 digitalWrite(T_Pin,HIGH);
 delayMicroseconds(10);
```

```
digitalWrite(T_Pin,LOW);
return pulseIn(E_Pin,HIGH)/58.2;
```

Check the results in the *Serial monitor*.

What's new:

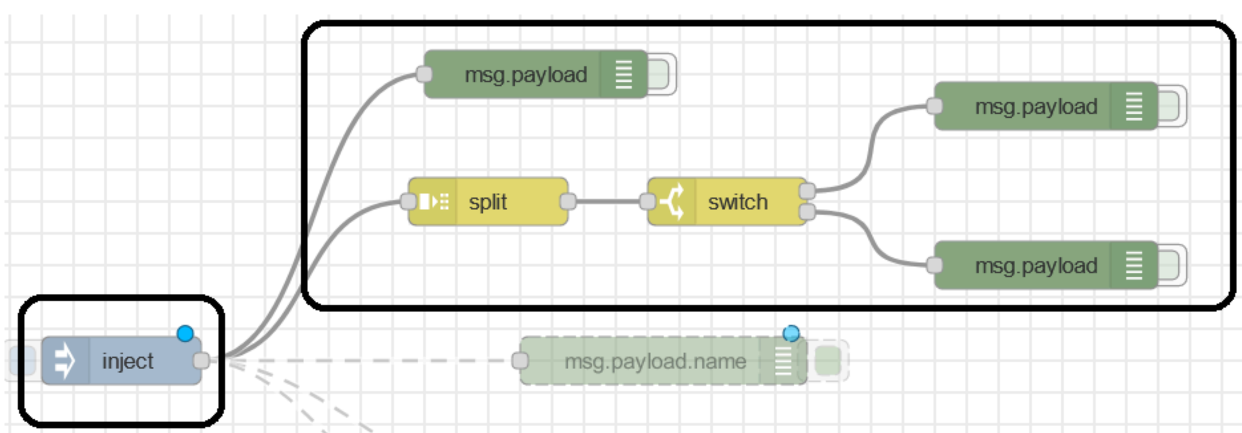
`StaticJsonDocument` object, `serializeJson` function

Task.8. Create a Node-Red dashboard to present distance measurement results.

The procedure for starting the Node-RED environment is as follows:

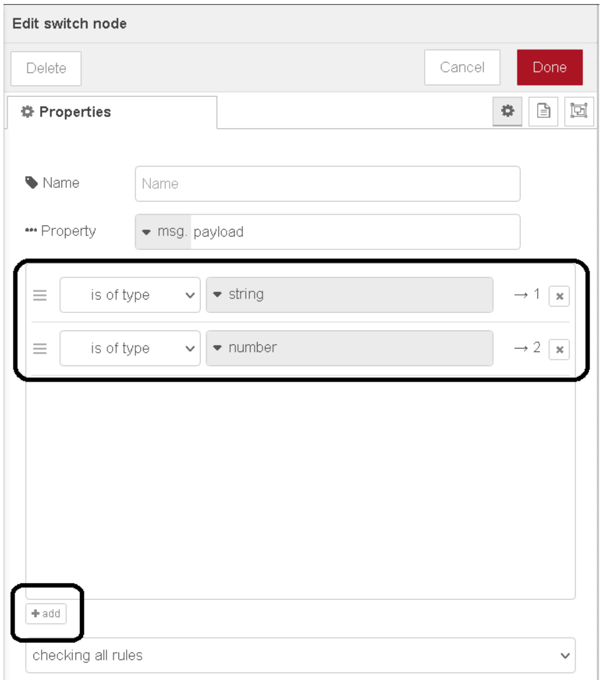
- launch the command line/Terminal;
- execute the `node-red` command;
- launch a web browser of Your choice;
- type `127.0.0.1:1880` in the address bar. Alternatively, you can enter `localhost:1880`.

Implement the indicated parts of the flow diagram.

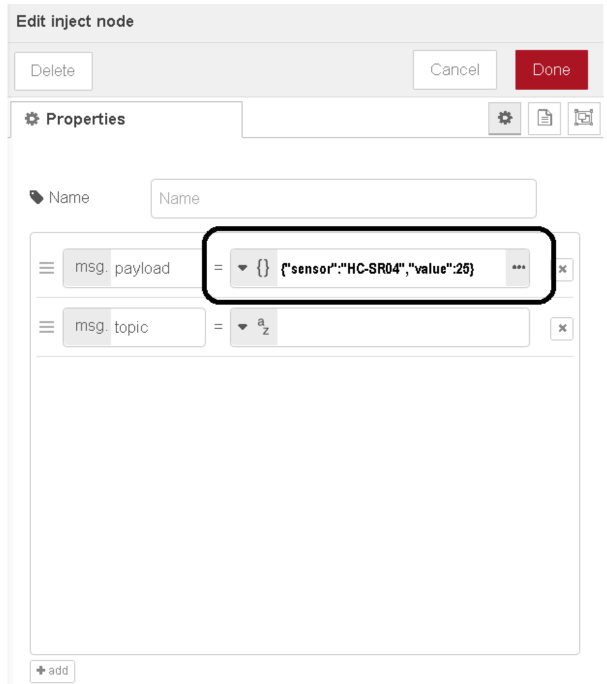


The *switch* node and the *inject* node configurations are presented below.

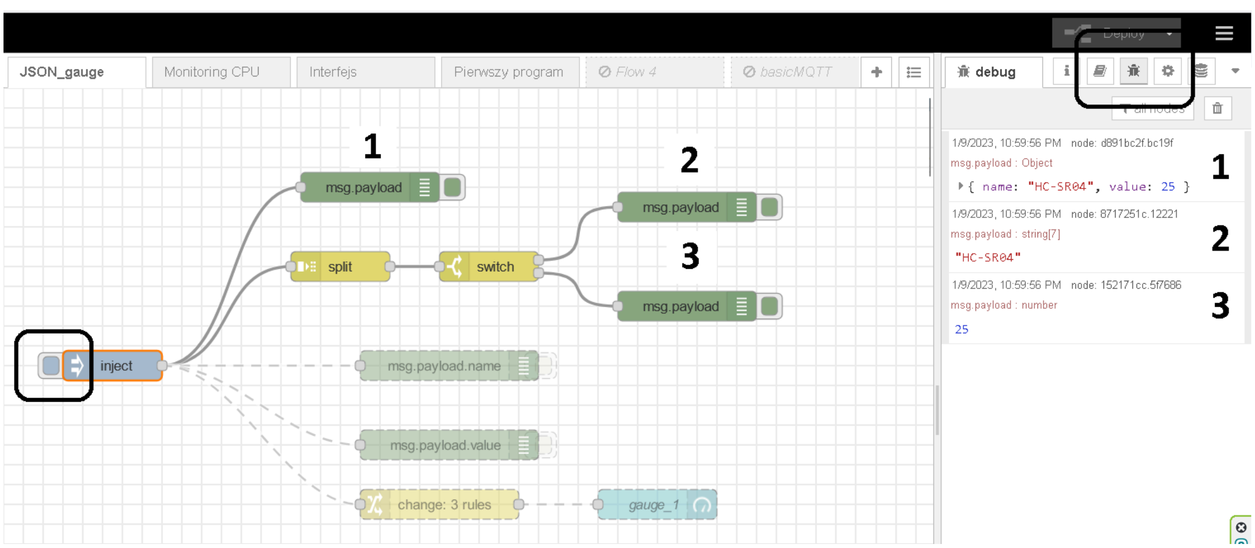
switch node configuration



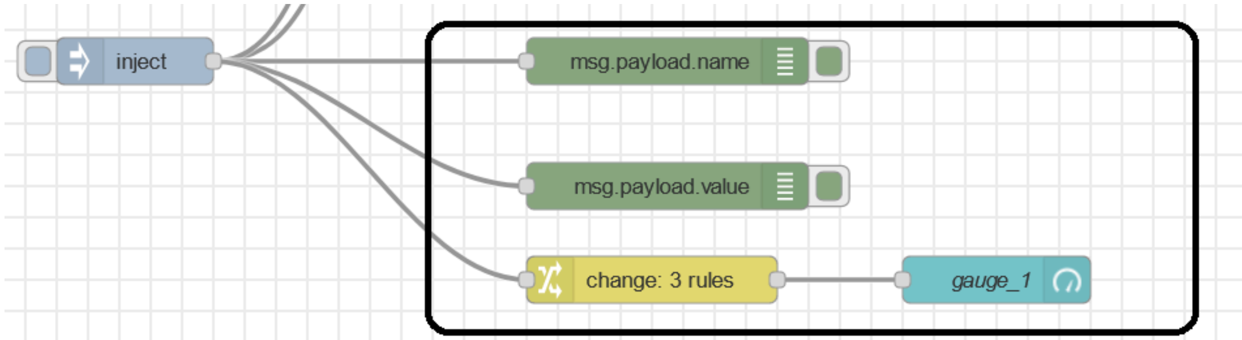
inject node configuration



Press the ***Deploy*** button. Trigger the *inject* node with the button on the left side of the block. Observe results in the *Debug* window.

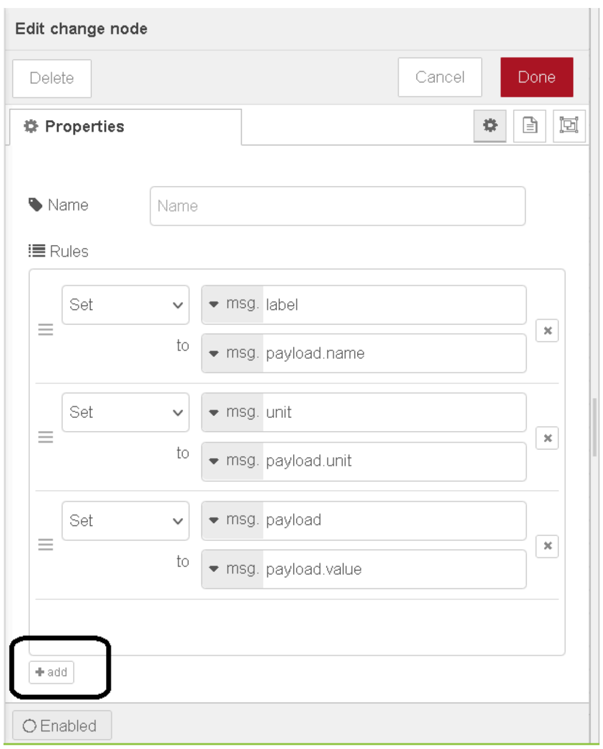


Implement the rest of the following flow diagram.

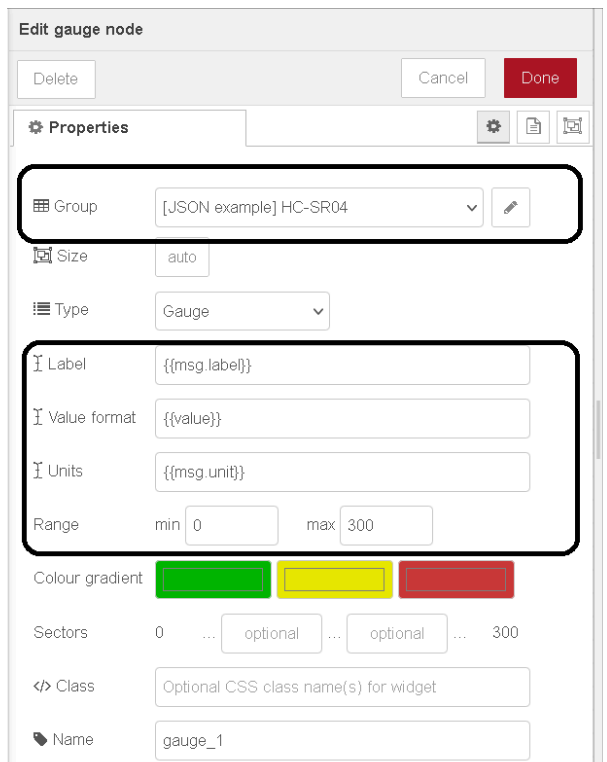


The *change* node and the *gauge* node configurations are presented below.

change node configuration



gauge node configuration



By default only one rule is available inside a *change* node. To add more rules press the *+add* button.

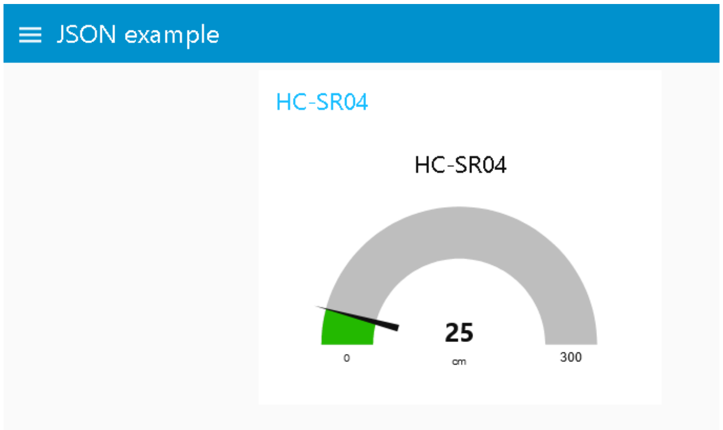
A *gauge* node allows for building a custom user interface. In the Node-RED environment, the interface is called the dashboard. Flows that allow You to build a dashboard are not part of a typical *Node-RED* package installation. They must be added separately using the options *Manage palette* (*Menu → Manage palette*). After selecting the *Manage palette* option, the *User Settings* window opens. The first step in the installation process is to switch to the *Install* tab. Then, in the search box, type *dashboard.* From the list of available options, select *node-red-dasboard* and press the *Install* button. After completing the installation process, press the *Close* button. We can check the correctness of the installation by scrolling the window with available blocks to the *dashboard* tab.

Each object on the interface must be assigned to a tab (*Tab*) and a group (*Group*). If You do not have defined groups and tabs in the newly built diagram, so with the help of *Add new dashboard group...* enter the name of the group as *HC-SR04*. A tab is entered with *Add new dashboard tab...* Call it as You wish eg. *JSON example*. Confirm with the *Add* button - 2 times. When the *Done* button is pressed, the configuration of the *Button* control is complete. Change *Label* and *Unis* properties according to `{{msg.label}}` and `{{msg.unit}}` respectively.

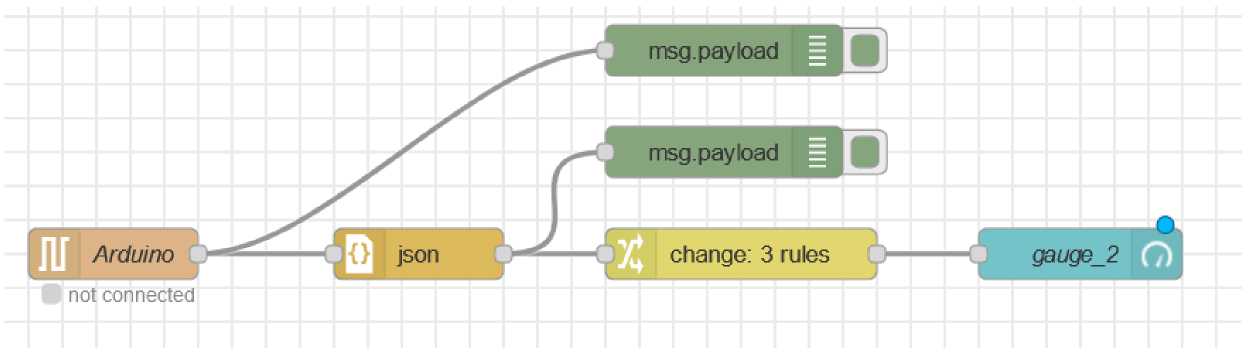
Modify the JSON object inside the *inject* node:

```
{ “name” : ”HC-SR04”, “value” : 0.0, “units” : ”cm” }
```

Press the ***Deploy*** button. Open a new window in a web browser. Type `127.0.0.1:1880`**`/ui`** in the address bar. Alternatively, you can enter `localhost:1880`**`/ui`**.



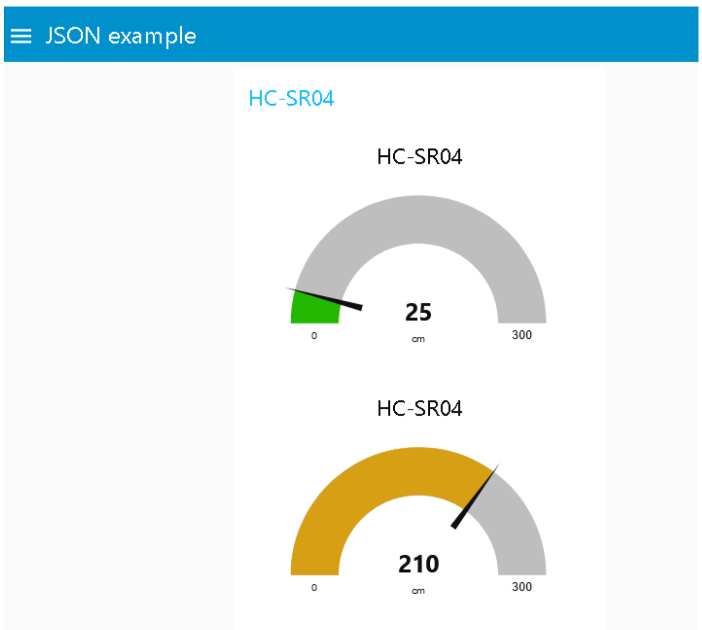
Add the *serial in* node. Configure connection parameters as we did in exercise no 5. Change the name of this node to Arduino. The *json* node is available in the *parser* palette.



For Arduino use the solution of task no 7. Modify the JSON string by adding a third key-value pair. Note that the capacity of the JSON object must be increased by 1.

```
#include<ArduinoJson.h>

#define BAUDRATE 115200
#define TriggerPin 3
#define EchoPin 2
#define MAX_RANGE 280

const int capacity = JSON_OBJECT_SIZE(3);
```

```
StaticJsonDocument<capacity> sensor;

void setup() {
 Serial.begin(BAUDRATE);
 sensor["name"] = "HC-SR04";
 sensor["value"] = 0.0;
 sensor["unit"] = "cm";
 pinMode(TriggerPin,OUTPUT);
 pinMode(EchoPin,INPUT_PULLUP);
}
```



Task.9. Use imperial units to display distance measurement results.

Task.10. Connect LM-35 temperature sensor (ex. no 3). Display ambient temperature in degrees of Celsius, degrees of Fahrenheit, and Kelvins on the dashboard. Add charts to present past results.

For those interested:

1. HowToMechatronics tutorial.
   howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/

2. Random Nerd Tutorials.

    randomnerdtutorials.com/complete-guide-for-ultrasonic-sensorhc-sr04/

3. Libraries contributed by the Arduino community.
    - www.arduino.cc/reference/en/libraries/hc-sr04/
    - www.arduino.cc/reference/en/libraries/hcsr04-ultrasonic-sensor/
    - www.arduino.cc/reference/en/libraries/hcsr04/
    - www.arduino.cc/reference/en/libraries/hc_sr04/

4. Introducing JSON.

    www.json.org/json-en.html

5. Arduino JSON documentation.

    arduinojson.org/v6/doc/

6. Add libraries to Arduino IDE.

    support.arduino.cc/hc/en-us/articles/5145457742236-Add-libraries-to-Arduino-IDE