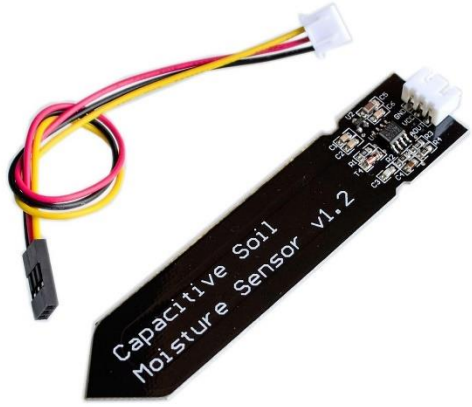
IRRIGATION UNIT

Sensors used:

LIQUID DETECTION: **XKC-Y25-NPN**
Link: http://www.icstation.com/contact-liquid-level-sensor-ip67-waterproof-output-water-level-detector-p-12292.html

WATER LEVEL: **JSN-SR04T**
Link: https://www.openelectronics.eu/Vandeniui-atsparus-ultragarsinis-atstumo-jutiklis-JSN-SR04T-AJ-SR04M

SOIL MOISTURE:
Link: https://protosupplies.com/product/capacitive-soil-moisture-sensor-module/



Source code - cloud:

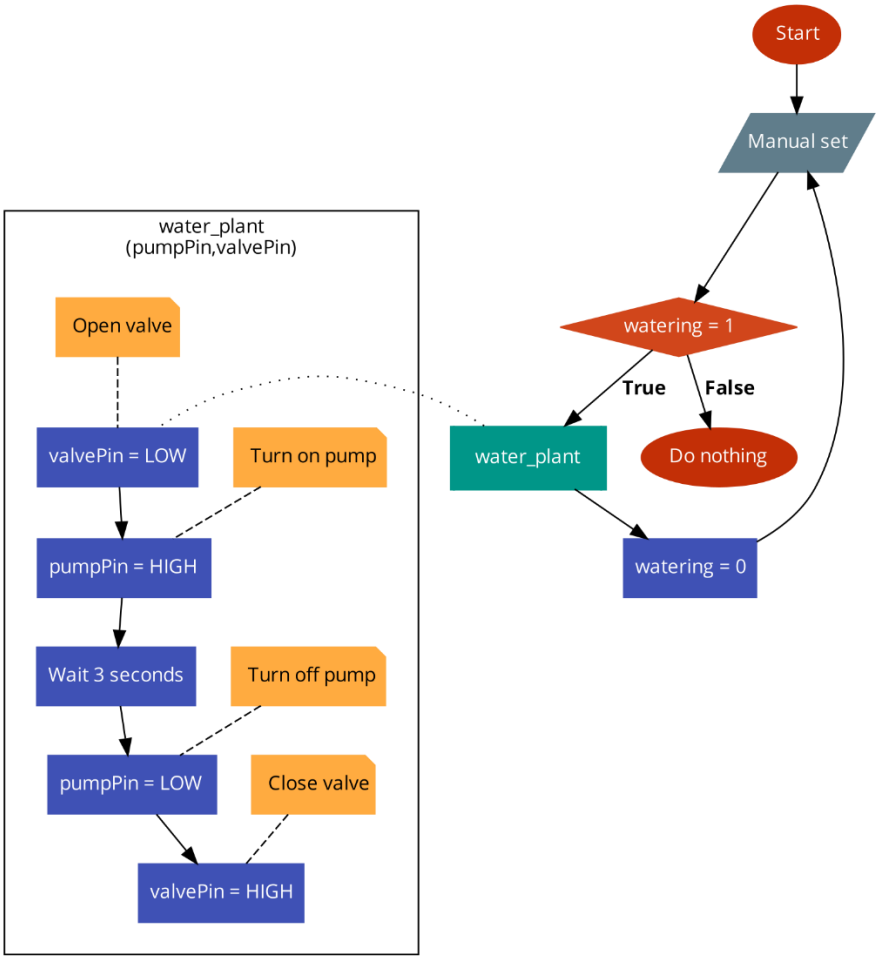
```
/*
  Sketch generated by the Arduino IoT Cloud Thing "Untitled"
  https://create.arduino.cc/cloud/things/2d441620-03b7-4c7e-
b69c-3a5f32006649

  Arduino IoT Cloud Variables description

  The following variables are automatically generated and
updated when changes are made to the Thing

  float liquid_level;
  float relative_moisture;
  float relative_moisture_2;
  int liquid_detection;
  int soil_moisture;
  int soil_moisture_2;
  int watering;
  int watering_2;
  bool switchPump;
  bool valve;

  Variables which are marked as READ/WRITE in the Cloud Thing
will also have functions
  which are called when their values are changed from the
Dashboard.
  These functions are generated with the Thing and added at
the end of this sketch.
*/

#include "thingProperties.h"

//Defining constants and variables
long duration;
int distance;
int pump;
const int sensor_distance = 60;
const int AirValue = 2512;
const int WaterValue = 1230;
const int wateringTime = 3;

#define moisturePin 32
#define moisturePin_2 33
#define liquid_detectionPin 4
#define triggerPin 14
#define echoPin 27
#define pumpPin 25
#define valvePin 17
#define pumpPin_2 26
```

```
#define valvePin_2 16

//Function to call during watering
void water_plant(int poumppin, int valvepin) {
  digitalWrite(valvepin,LOW);
  digitalWrite(poumppin,HIGH);
  delay(3000);
  digitalWrite(poumppin,LOW);
  digitalWrite(valvepin,HIGH);
}

void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  // This delay gives the chance to wait for a Serial Monitor
without blocking if none is found
  delay(1500);


  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);

  /*
     The following function allows you to obtain more
information
     related to the state of network and IoT Cloud connection
and errors
     the higher number the more granular information you'll
get.
     The default is 0 (only errors).
     Maximum is 4
  */
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();


  pinMode(liquid_detectionPin, INPUT); // Set pin for liquid
detection as input pin
  pinMode(moisturePin, INPUT); // Set pin for moisture
measurements as input pin
  pinMode(valvePin, OUTPUT);    // Set pin for valve control
as output pin
  pinMode(pumpPin, OUTPUT);       // Set pin for pump control
as output pin
  pinMode(triggerPin, OUTPUT); // configure the triggerPin(D9)
as an Output
  pinMode(echoPin, INPUT); // configure the echoPin(D11) as an
Input
```

```
  digitalWrite(pumpPin, LOW);    // Turn off pump
  digitalWrite(valvePin, HIGH);    // Close valve

  pinMode(moisturePin_2, INPUT); // Set pin for moisture
measurements as input pin
  pinMode(valvePin_2, OUTPUT); // Set pin for valve control as
output pin
  pinMode(pumpPin_2, OUTPUT); // Set pin for pump control as
output pin
  digitalWrite(pumpPin_2, LOW);    // Turn off pump
  digitalWrite(valvePin_2, HIGH);    // Close valve
}

void loop() {
  ArduinoCloud.update();
  // Inserted code


  //LIQUID DETECTION:
  liquid_detection = digitalRead(liquid_detectionPin);
  Serial.print(liquid_detection); //Digital value: 1 (liquid
detected) or 0 (no liquid)
  Serial.print(": ");
  if (digitalRead(liquid_detectionPin)) {
    Serial.println("Liquid Detected!");
  }
  else {
    Serial.println("No Liquid!");
  }
  delay(500);

  //SOIL_MOISTURE
  relative_moisture = analogRead(moisturePin);
  //Maps a number from one range to another:
  //map(value, fromLow, fromHigh,  toLow, toHigh)
  soil_moisture = map(relative_moisture, AirValue, WaterValue,
0, 100);
  if (soil_moisture >= 100) {
    soil_moisture = 100;
  } else if (soil_moisture <= 0) {
    soil_moisture = 0;
  }
  Serial.print("Relative: ");
  Serial.println(relative_moisture);
  Serial.print("Soil: ");
  Serial.println(soil_moisture);
  delay(2);

  relative_moisture_2 = analogRead(moisturePin_2);
  soil_moisture_2 = map(relative_moisture_2, AirValue,
WaterValue, 0, 100);
```

```
  if (soil_moisture_2 >= 100) {
    soil_moisture_2 = 100;
  } else if (soil_moisture_2 <= 0) {
    soil_moisture_2 = 0;
  }
  Serial.print("Relative: ");
  Serial.println(relative_moisture_2);
  Serial.print("Soil: ");
  Serial.println(soil_moisture_2);
  delay(2);


  //WATERING
  if (watering == 1) {             // Chceck if needs watering
    water_plant(pumpPin, valvePin);
    watering = 0;                   // Update watering state
  }

  if (watering_2 == 1) {           // Chceck if needs watering
    water_plant(pumpPin_2, valvePin_2);
    watering_2 = 0;                   // Update watering state
  }


  //WATER LEVEL
  digitalWrite(triggerPin, LOW); //set trigger signal low for
2us
  delayMicroseconds(2);

  /*send 10 microsecond pulse to trigger pin of HC-SR04 */
  digitalWrite(triggerPin, HIGH); // make trigger pin active
high
  delayMicroseconds(10);           // wait for 10 microseconds
  digitalWrite(triggerPin, LOW); // make trigger pin active
low

  /*Measure the Echo output signal duration or pulse width */
  duration = pulseIn(echoPin, HIGH); // save time duration
value in "duration variable
  distance = duration * 0.034 / 2; //Convert pulse duration
into distance
  liquid_level = sensor_distance - distance;

  // print measured distance value on Arduino serial monitor
  Serial.print("Water level: ");
  Serial.print(liquid_level);
  Serial.println(" cm");
  delay(1000);
}
```

```
/*
  Since Valve is READ_WRITE variable, onValveChange() is
  executed every time a new value is received from IoT Cloud.
*/
void onValveChange()  {
// Switch between the two valves
    if (valve)
      digitalWrite(valvePin, HIGH);
    else
      digitalWrite(valvePin, LOW);
}

/*
  Since SwitchPump is READ_WRITE variable,
onSwitchPumpChange() is
  executed every time a new value is received from IoT Cloud.
*/
void onSwitchPumpChange()  {
  // Switch between the two pumps
  if(switchPump)
    digitalWrite(pumpPin, HIGH);
  else
    digitalWrite(pumpPin, LOW);
}
```



SENSOR MODULE

Sensors used:

eCO_2 and TVOC measurements

Sensor: **SGP30**
Link: https://www.adafruit.com/product/3709
Library: Adafruit SGP30

CO2 concentration

Sensor: MHZ-19b
Link: https://www.winsen-sensor.com/d/files/infrared-gas-sensor/mh-z19b-co2-ver1_0.pdf
Library: MH-Z19

Temperature and humidity measurements

Sensor: Si7021
Link: https://www.sonoff.sk/kategoria/komponenty-cidla-a-senzory/th-sensor-si7021-senzor-teploty-a-vlhkosti/
Library: DHTNEW

Source code:

```
//include libraries
#include <Arduino.h>
#include <Wire.h>
#include "Adafruit_SGP30.h"
#include <dhtnew.h>
#include "MHZ19.h"
#include <SoftwareSerial.h>

//pin definition
#define RX_PIN 16          // Rx pin which the MHZ19 Tx pin is attached to
#define TX_PIN 17         // Tx pin which the MHZ19 Rx pin is attached to
#define BAUDRATE 9600  // Device to MH-Z19 Serial baudrate (should not be changed)

// define variables
int CO2;
float temperature;
float humidity;

// SGP30
Adafruit_SGP30 sgp;

uint32_t getAbsoluteHumidity(float temperature, float humidity) {
//approximation formula from Sensirion SGP30 Driver Integration chapter 3.15
```

```
const float absoluteHumidity = 216.7f * ((humidity / 100.0f) *
6.112f * exp((17.62f * temperature) / (243.12f + temperature))
/ (273.15f + temperature)); //[g/m^3]
const uint32_t absoluteHumidityScaled =
static_cast<uint32_t>(1000.0f * absoluteHumidity); //[mg/m^3]
return absoluteHumidityScaled;
}

// Si7021
DHTNEW mySensor(4);

// MHZ-19b
MHZ19 myMHZ19;              // Constructor for library
SoftwareSerial mySerial(RX_PIN, TX_PIN);  // Create device to
MH-Z19 serial

void setup() {
  Serial.begin(115200);
  while (!Serial) { delay(10); }  // Wait for serial console
to open!

  mySerial.begin(BAUDRATE);  // (Uno example) device to MH-Z19
serial start
  myMHZ19.begin(mySerial);      // *Serial(Stream) refence
must be passed to library begin().

  myMHZ19.autoCalibration();  // Turn auto calibration ON (OFF
autoCalibration(false))

// SGP30
  Serial.println("SGP30 test");

  if (! sgp.begin()){
    Serial.println("Sensor not found :(");
    while (1);
  }
  Serial.print("Found SGP30 serial #");
  Serial.print(sgp.serialnumber[0], HEX);
  Serial.print(sgp.serialnumber[1], HEX);
  Serial.println(sgp.serialnumber[2], HEX);

  // If you have a baseline measurement from before you can
assign it to start, to 'self-calibrate'
  //sgp.setIAQBaseline(0x8E68, 0x8F41);  // Will vary for each
sensor!

// temp. and humidity measurements from Si7021 are used in
SGP30 code
  mySensor.setHumOffset(10);
  mySensor.setTempOffset(-3.5);
}
```

```
int counter = 0;
void loop() {
mySensor.read();

temperature = mySensor.getTemperature();
humidity = mySensor.getHumidity();
// If you have a temperature / humidity sensor, you can set the
absolute humidity to enable the humditiy compensation for the
air quality signals
sgp.setHumidity(getAbsoluteHumidity(temperature, humidity));

if (! sgp.IAQmeasure()) {
    Serial.println("Measurement failed");
    return;
 }

Serial.print("Temperature    ");    Serial.print(temperature);
Serial.print(" C\t");
Serial.print("Humidity       ");       Serial.print(humidity);
Serial.print(" %\t");
Serial.print("TVOC  "); Serial.print(sgp.TVOC); Serial.print("
ppb\t");
Serial.print("eCO2  "); Serial.print(sgp.eCO2); Serial.print("
ppm\t");

if (! sgp.IAQmeasureRaw()) {
    Serial.println("Raw Measurement failed");
    return;
 }
// MHZ-19b measurement
CO2 = myMHZ19.getCO2();
Serial.print("CO2 - mhz "); Serial.print(CO2); Serial.println("
ppm");
delay(2000);

  counter++;
  if (counter == 30) {
    counter = 0;

    uint16_t TVOC_base, eCO2_base;
    if (! sgp.getIAQBaseline(&eCO2_base, &TVOC_base)) {
      Serial.println("Failed to get baseline readings");
      return;
    }
    Serial.print("****Baseline     values:     eCO2:     0x");
Serial.print(eCO2_base, HEX);
    Serial.print("  &  TVOC:  0x");  Serial.println(TVOC_base,
HEX);
  }
}
```