Data Farming

Christopher Moran, Solutions Architect, Ramsay Healthcare

https://blog.nowwhywouldyoudothat.com

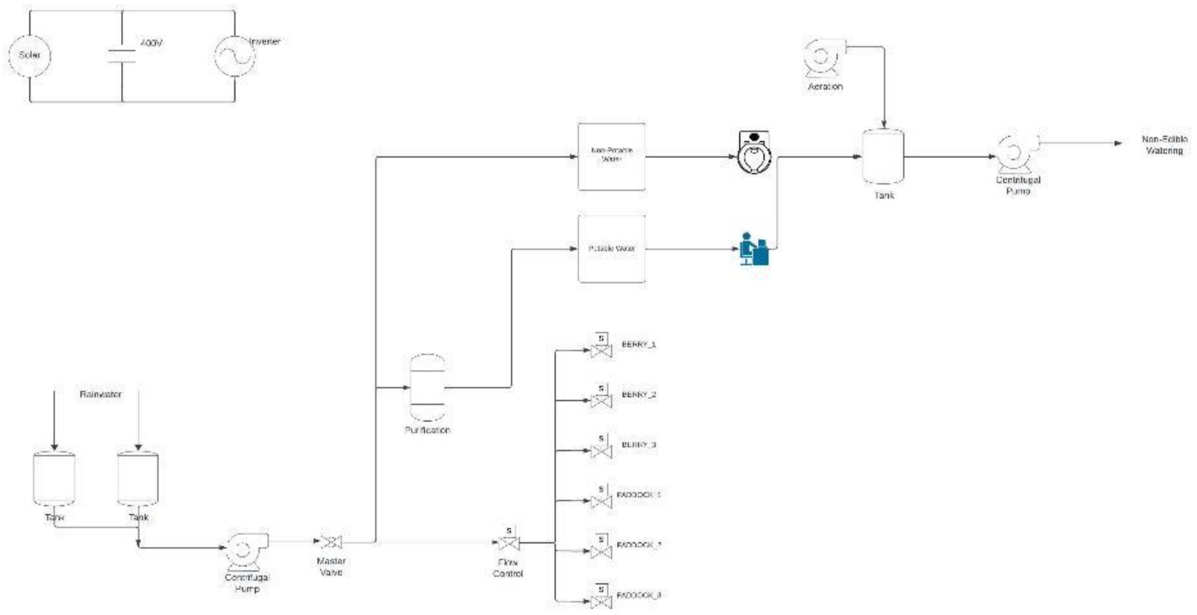
Legal made me do it…

- ✓ The following presentation is the result of a private research project and is not sponsored or endorsed by my employer, Ramsay Health Care. While this research project has fed into some of my professional work, this is entirely my own creation, and any intellectual property is mine to use as I see fit.
- ✓ A lot of product names and brands are mentioned in this presentation. Any mention does not constitute my endorsement of that company or said company's endorsements of my activities.
- ✓ I have released all source code for this research under a standard BSD "3-Clause" license, and if you use any of this work then you agree to be bound by the terms of that license.
- ✓ There. That ought to do it.

Problem Definition - 1

- ✓ Background
  - ✓ We live on a “larger than usual” semi-rural property on the outskirts of Sydney
  - ✓ We don’t have town water, sewerage or electricity
  - ✓ We like our creature comforts (Aircon, hot water, TV, internet, lights in the nighttime)
  - ✓ My wife likes to garden and has a lot of plants
  - ✓ I love data
- ✓ Problem
  - ✓ How much water do we have?
  - ✓ Are the batteries charged?
  - ✓ Is that ^#*$&* septic tank about to overflow?

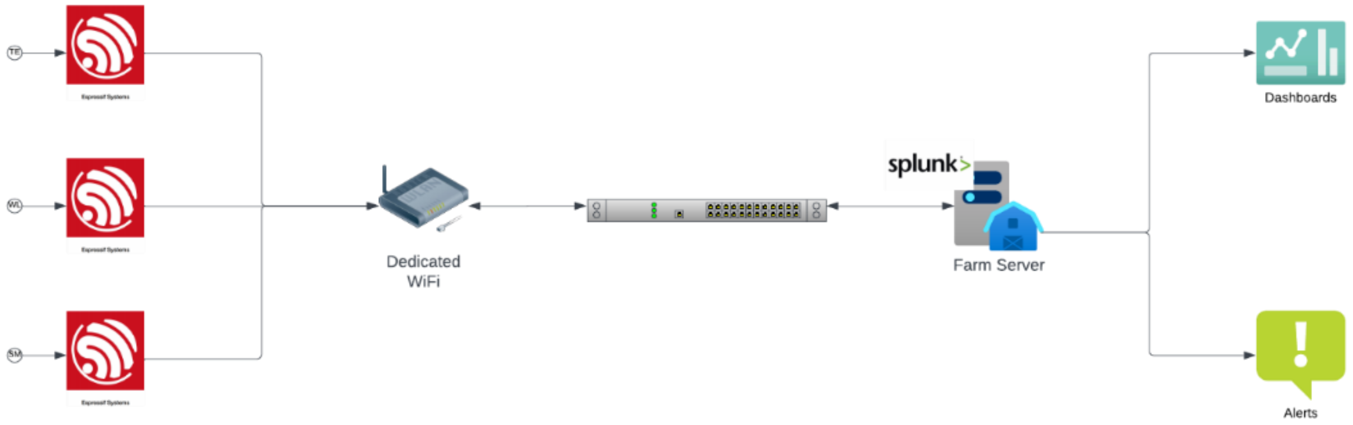
Problem Definition - 2



Seriously?

- ✓ I need to monitor everything
  - ✓ Rainwater
    - ✓ Tank depth, Water Temperature, Pump State
  - ✓ Power
    - ✓ Solar output, Battery Input & Output, Inverter State
  - ✓ Gardens
    - ✓ Soil Moisture, Water Flow, Temperature
  - ✓ Climate
  - ✓ Where’s my mower?

The Solution



DIY Monitor Hardware

Wi Fi™

ARDUINO™

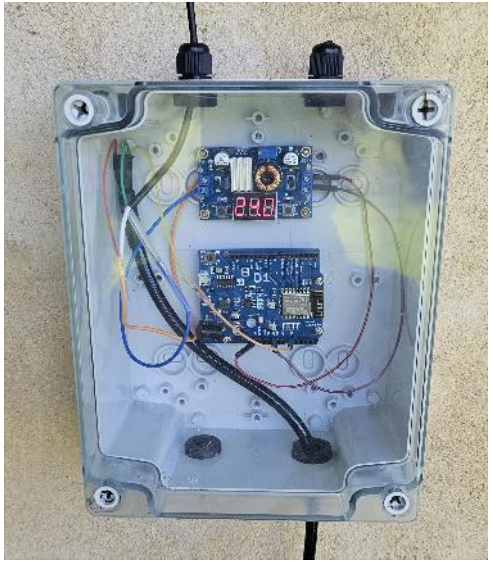
Espressif Systems

BUNNINGS warehouse

jaycar

JACK DANIEL'S OLD TIME Old No.7 BRAND

DIY Monitor Hardware



Sensor Details

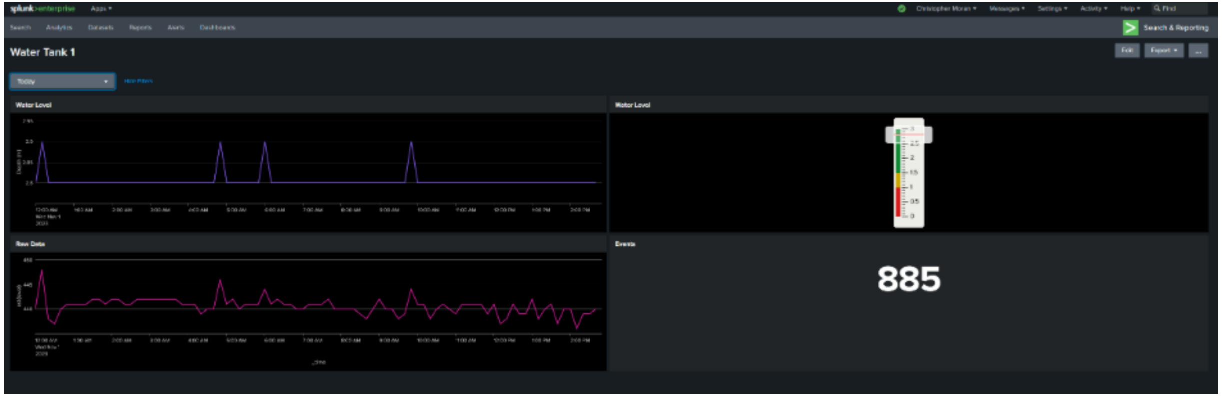
- ✓ ESP-32 WROOM modules
  - ✓ Cheap
  - ✓ Pretty easy to use
  - ✓ Not very reliable – Buy spares!
  - ✓ Can use ESP8266 in some cases
- ✓ Hydrostatic pressure sensor for tank depth
  - ✓ Because ultrasonics, floats and lasers will make you crazy
  - ✓ Of course, the outputs aren’t compatible with the ESP-32
- ✓ Current transformers for power monitoring
  - ✓ Needs signal conditioning
- ✓ Bosch BMP280 environmental sensor
- ✓ Capacitive soil moisture sensors
  - ✓ Electrolysis isn’t nice
- ✓ Fronius Inverters have a Python interface
- ✓ Duracell have a web API, but it’s under NDA

Why Splunk?

- ✓ It’s made to store, analyse and present data
- ✓ I already know how to use it
- ✓ My needs fall within the “free” use case. Even with 110 sensors
- ✓ Full disclosure – I did try:
  - ✓ ELK – Works, but visualising gets messy
  - ✓ Grafana/Prometheus – DIY is big hardware; Cloud can get expensive
  - ✓ AWS IoT – No inbuilt data retention without extra cost

How?

- ✓ Create an index so you know how much data you’re using.
- ✓ HTTP Collector is easy to use.
- ✓ Have the input timestamp the data so that you don’t have to.
- ✓ Write some dashboards, or better, get an artistic person to write the dashboards.



Demo Time

What did I learn?

- ✓ You can do a lot on a limited budget.
- ✓ Document what you have before you start building.
- ✓ Analogue inputs on ESP-32s are crap.
- ✓ Your dog will dig up any, and all cables you bury.
- ✓ IP65 doesn’t mean waterproof if you put holes in it.
- ✓ Monitor battery voltage on remote sensor and alert when low.
- ✓ Don’t do unit conversions on the sensors. Let Splunk handle it.
- ✓ Your neighbours and significant other won’t understand your greatness.
- ✓ There is always more data.

Thank You!

https://github.com/au-chrismor/datafarm

https://blog.nowwhywouldyoudothat.com

Source Types

```
{
    “host”: “sensor_name”,
    “sourcetype”: “datafarm”,
    “index”: “index_name”,
    “event”: {
        “sensortype”: “sensor_type”,
        specific-data-pairs
    }
}
```

Sensortype = “tankdepth”

```
{
     “host”: “sensor_name”,
     “sourcetype”: “datafarm”,
     “index”: “index_name”,
     “event”: {
          “sensortype”: “tankdepth”,
          “depth”: ADC Value (int),
          “temperature”: DS18B20 Output (float),
          “battery”: ADC Value (int)
     }
}
```

Set to “-274” if not used

Set to “-1” if not used

Sensortype = “soilmoisture”

```
{
        “host”: “sensor_name”,
        “sourcetype”: “datafarm”,
        “index”: “index_name”,
        “event”: {
                “sensortype”: “soilmoisture”,
                “moisture”: ADC Value (int),
                “battery”: ADC Value (int)
        }
}
```

Set to “-1” if not used

Sensortype = “temperature”

```
{
    “host”: “sensor_name”,
    “sourcetype”: “datafarm”,
    “index”: “index_name”,
    “event”: {
        “sensortype”: “temperature”,
        “temperature”: DS18B20 Output (float),
        “battery”: ADC Value (int)
    }
}
```

Set to “-1” if not used

Sensortype = “environment”

```
{
        “host”: “sensor_name”,
        “sourcetype”: “datafarm”,
        “index”: “index_name”,
        “event”: {
              “sensortype”: “environment”,
              “temperature”: BME280 Value (float),
              “humidity”: BME280 Value (float),
              “pressure”: BME280 Value (float),
              “battery”: ADC Value (int)
        }
}
```

Set to “-1” if not used

Sensortype = “acpower”

```
{
    “host”: “sensor_name”,
    “sourcetype”: “datafarm”,
    “index”: “index_name”,
    “event”: {
        “sensortype”: “acpower”,
        “voltage”: ADC Value (int),
        “current”: ADC Value (int),
        “battery”: ADC Value (int)
    }
}
```

Set to “-1” if not used

Sensortype = “dcpower”

```
{
     “host”: “sensor_name”,
     “sourcetype”: “datafarm”,
     “index”: “index_name”,
     “event”: {
          “sensortype”: “dcpower”,
          “voltage”: ADC Value (int),
          “current”: ADC Value (int),
          “battery”: ADC Value (int)
     }
}
```

Set to “-1” if not used

Sensortype = “weather”

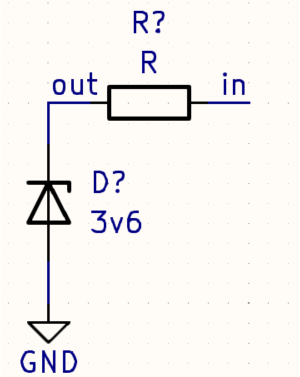
```
{
    “host”: “sensor_name”,
    “sourcetype”: “datafarm”,
    “index”: “index_name”,
    “event”: {
        “sensortype”: “weather”,
        “windspeed”: last m/s (int),
        “winddir”: lookup value (int),
        “rainfall”: pulse count (int),
        “uv”: ADC value (int),
        “light”: ADC value (int),
        “temperature”: BME280 Value (float),
        “humidity”: BME280 Value (float),
        “pressure”: BME280 Value (float),
        “battery”: ADC Value (int)
    }
}
```

Set to “-1” if not used

Progress Photos

In case you didn't know…

Input Protection



ADC Resolution

Device Family	Resolution (Bits)	Count Range	Resolution*
ESP8266 / AVR Arduino	10	0 - 1023	2.4mm
ESP32 / Due / MKR	12	0 – 4093	0.7mm
ADS1115 (External)	16	0 – 65535	0.06mm

*Assuming 5V = 3m Depth