Prometheus for Pets

Ronald McCollam
October 2025

Ruggles

Not actually a borg

Problem statement

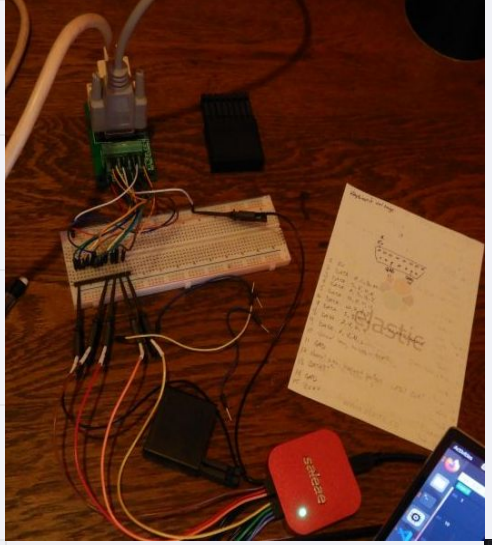
Could be: Keeping tabs on your pet: water consumption, outdoor time

Actually is: Having a fun project to experiment with IoT devices and data analytics

Disclaimer

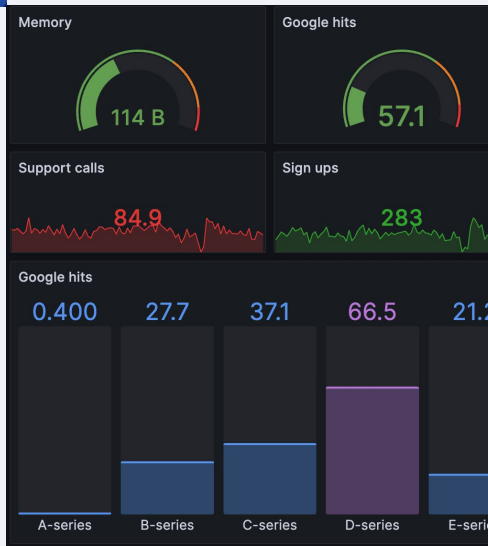
I’m not a veterinarian and this is not medical advice!

What we'll use



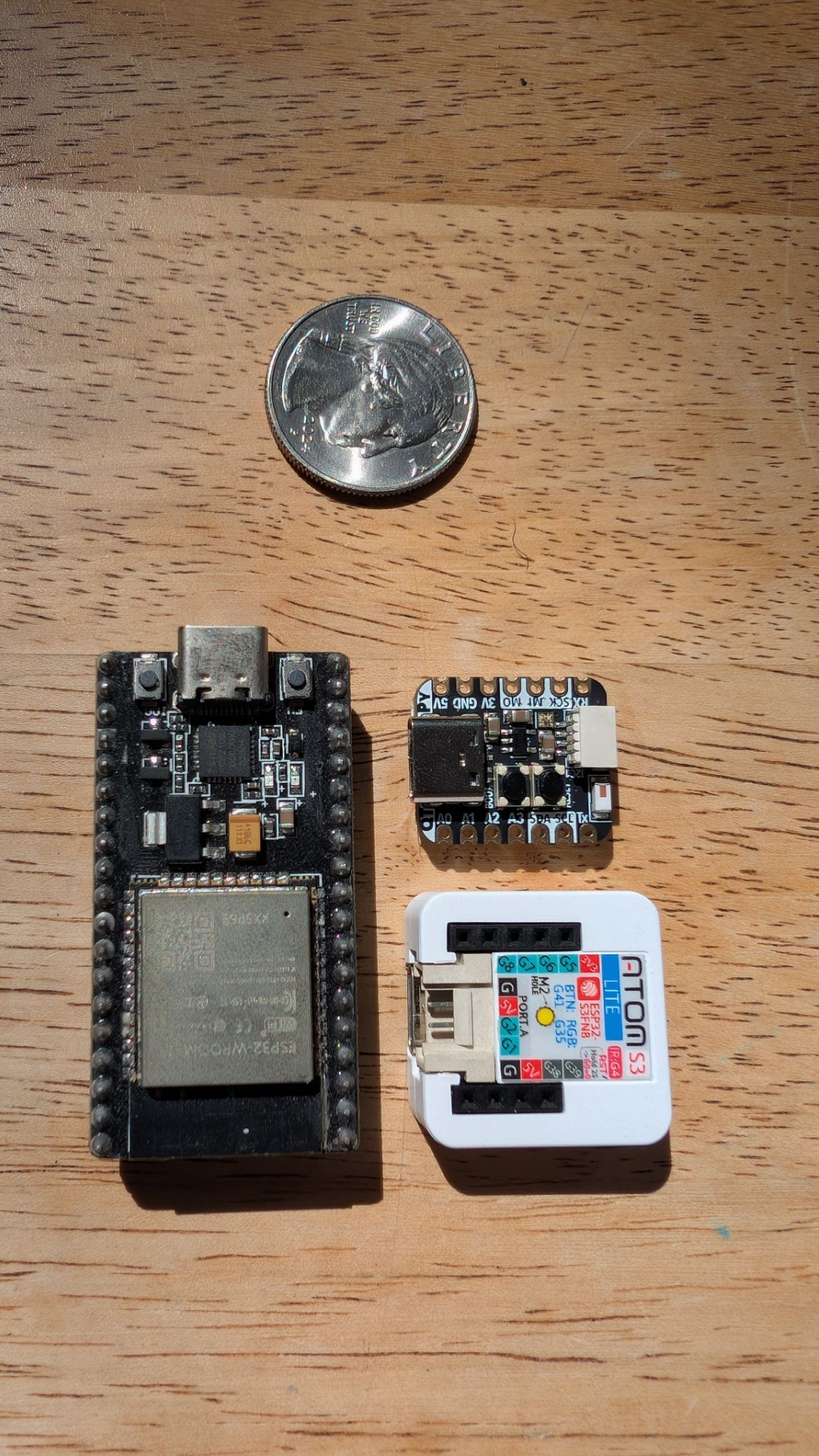
Hardware

- IoT dev boards based on the **ESP32**
- Off-the-shelf sensors (accelerometer, digital scale)
- Output devices (LEDs, speaker)
- A computer

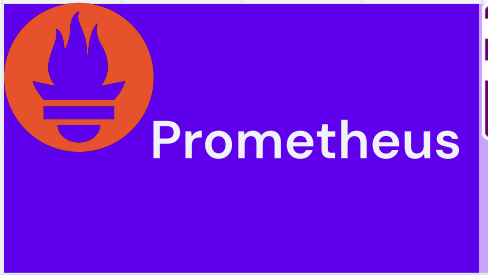


Software
(all open source)

- Prometheus
- MQTT (specifically *Mosquitto*)
- Grafana


ATOM S3
LITE
ESP32-S3FN8
BTN: G41
RGB: G35
M2 HOLE
PORT.A

Software



Data collector & time-series database

Prometheus has become the defacto metric system for the cloud computing world, used natively in Kubernetes (and many more)

Lightweight message queue / transport

A way of passing messages between services in constrained environments

(It stands for “Message Queuing Telemetry Transport”)

Flexible & extensible data visualization

Grafana is the “Swiss army chainsaw” of data visualization

What we'll build

Pet door monitor

Track entrances and exits of your pet

Water monitor

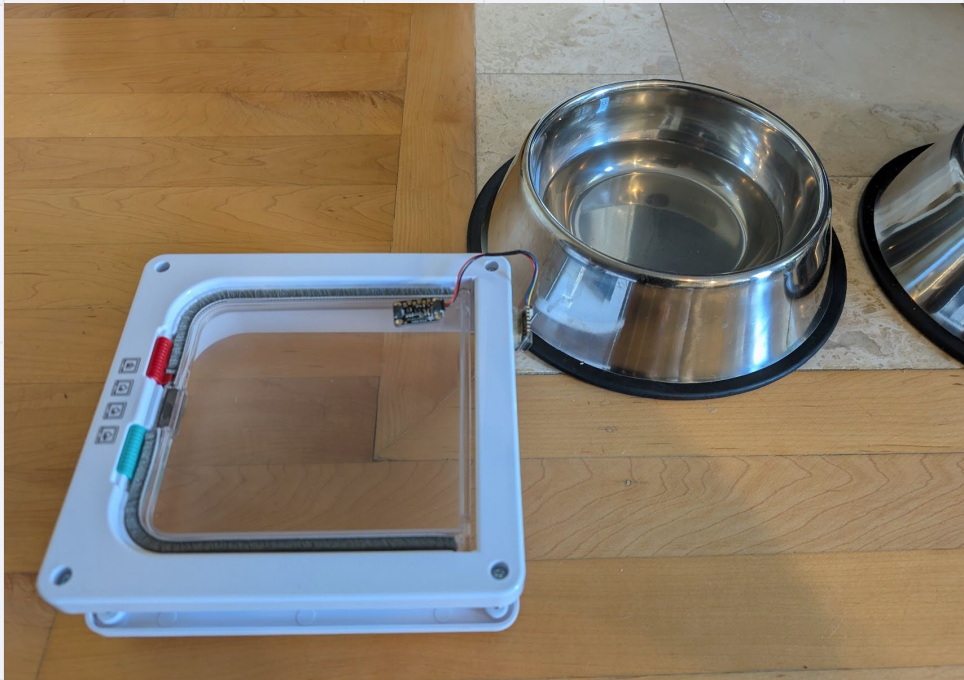
Check how much water is available and monitor its consumption

Alerting

Get notified when something is wrong (like your pet is running low on water)

Analytics

See trends over time and react when something changes



Prometheus

Database

Stores data collected

Query engine

Lets you ask questions about the data

UI

Lightweight web interface to query data

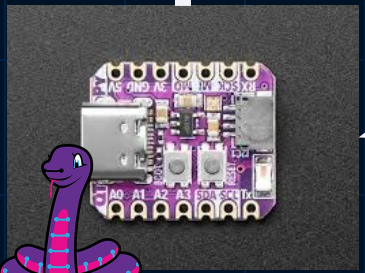
Collector

Gets data from whatever provides it

(Usually by “scraping”)

```
# HELP apt_upgrades_pending Apt package pending updates by origin.
# TYPE apt_upgrades_pending gauge
apt_upgrades_pending{arch="all",origin="Ubuntu:20.04/focal"} 2
apt_upgrades_pending{arch="all",origin="Ubuntu:20.04/focal-updates"} 12
apt_upgrades_pending{arch="amd64",origin="Ubuntu:20.04/focal-updates"} 36
# HELP node_cpu_seconds_total Seconds the cpus spent in each mode.
# TYPE node_cpu_seconds_total counter
node_cpu_seconds_total{cpu="0",mode="idle"} 7.21255975e+06
node_cpu_seconds_total{cpu="0",mode="iowait"} 14043.08
node_cpu_seconds_total{cpu="0",mode="irq"} 0
node_cpu_seconds_total{cpu="0",mode="nice"} 2241.3
node_cpu_seconds_total{cpu="0",mode="softirq"} 745148.43
node_cpu_seconds_total{cpu="0",mode="steal"} 0
node_cpu_seconds_total{cpu="0",mode="system"} 152842.28
node_cpu_seconds_total{cpu="0",mode="user"} 1.29785252e+06
# HELP node_disk_read_bytes_total The total number of bytes read successfully.
# TYPE node_disk_read_bytes_total counter
node_disk_read_bytes_total{device="sda"} 8.63297020928e+11
node_disk_read_bytes_total{device="sdb"} 6.77444059648e+12
node_disk_read_bytes_total{device="sdc"} 1.032704e+07
node_disk_read_bytes_total{device="sdd"} 0
node_disk_read_bytes_total{device="sde"} 0
node_disk_read_bytes_total{device="sr0"} 0
node_disk_read_bytes_total{device="sr1"} 0
```

Pet Door Sensor



Door Sensor

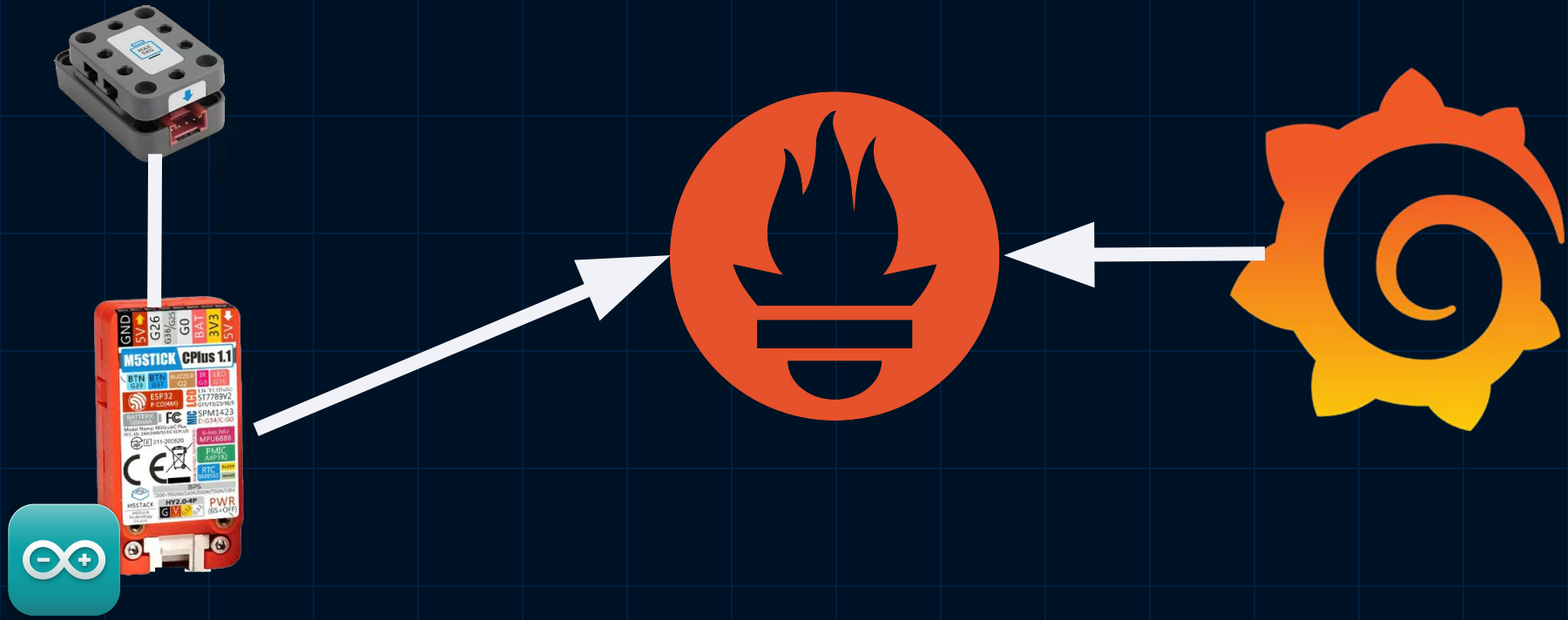
```
### Set up a server for Prometheus to scrape
pool = socketpool.SocketPool(wifi.radio)
server = Server(pool, debug=True)
server.start(str(wifi.radio.ipv4_address), port=8080)

# Send Prometheus data when requested
@server.route("/metrics")
def metrics_handler(request: Request):
  global entries, exits

  data = (
       '# HELP door_use_counter number of uses of the pet door\n'
    +  '# TYPE door_use_counter counter\n'
    + f'door_use_counter_total{{door="front_dog_door", type="entries"}} {entries}\n'
    + f'door_use_counter_total{{door="front_dog_door", type="exits"}} {exits}\n'
  )

  return Response(request, data, content_type="text/plain; version=0.0.4")
```

Door Sensor

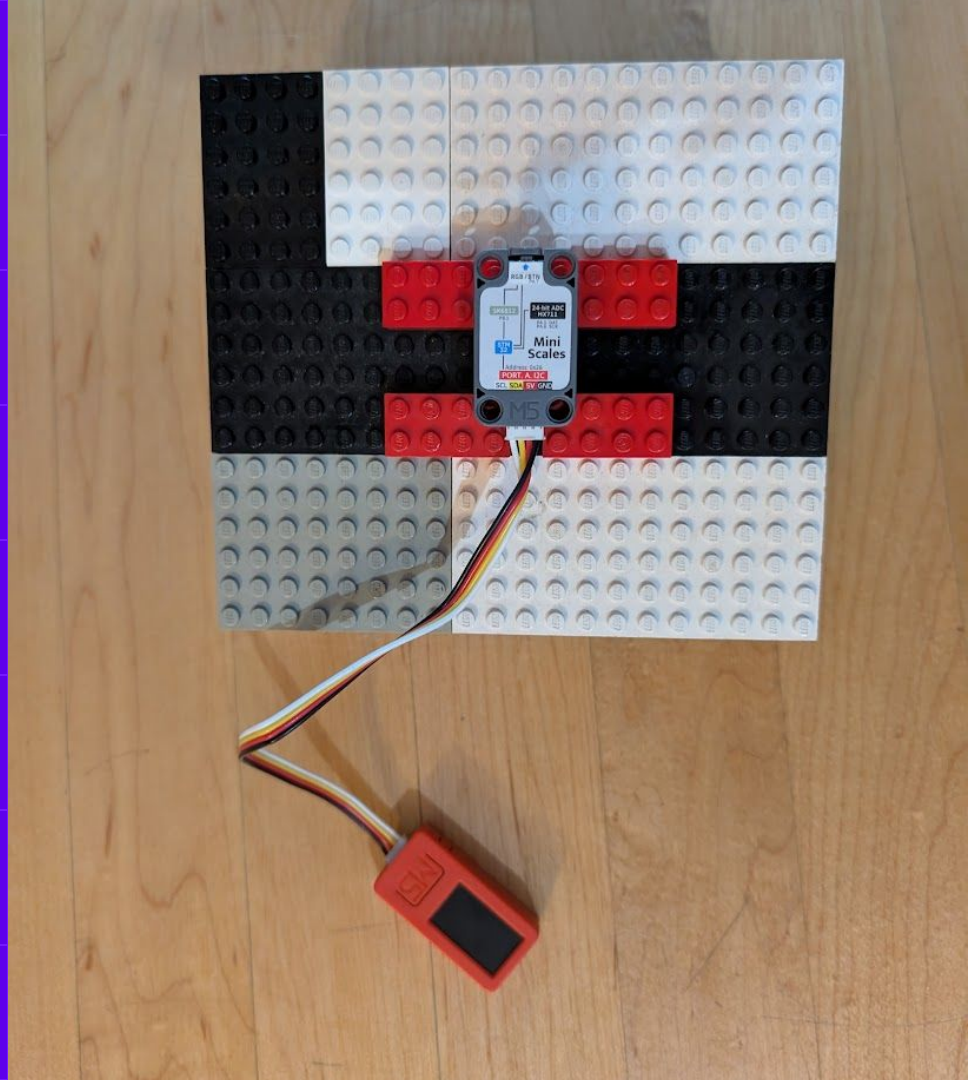
```
while True:
    # Look for uses of the pet door
    x, y, z = msa.acceleration
    if z > 4: # Door is in 'entry' position
        if prev_state != "entry":
            entries = entries + 1
            prev_state = "entry"
    elif z < -4: # Door is in 'exit' position
        if prev_state != "exit":
            exits = exits + 1
            prev_state = "exit"
    else: # Door is closed
        prev_state = "closed"

    # Serve the data to Prometheus
    server.poll()


    # Sleep a bit - prevents a tight loop in the webserver and also
    # acts as a cheap 'debounce' so we don't see a bunch of data when it's right on the margin
    time.sleep(0.25)
```

DEMO

Water Sensor



Now you know why



Water Sensor

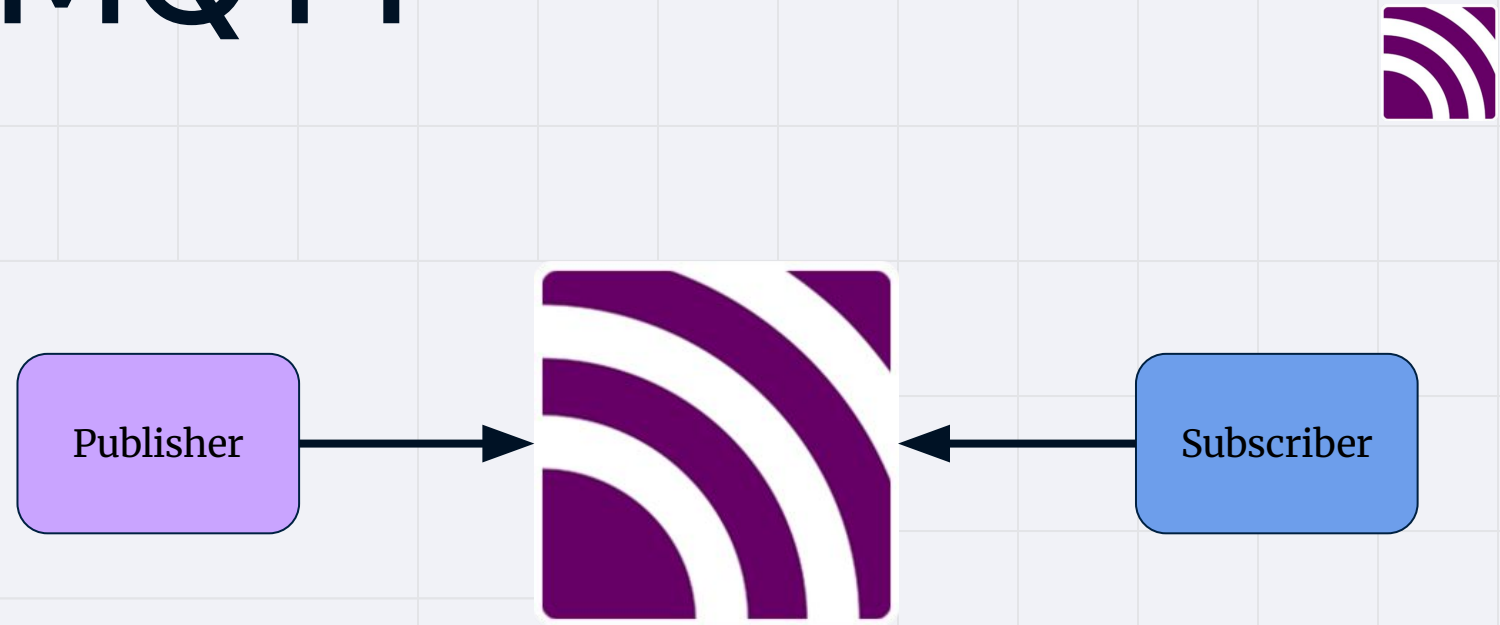
```
TimeSeries ts_weight(5, "weight_grams", "{job=\"pet_monitor\",device=\"water_scale\"}");

void setup() {
    /*** Set up the hardware ***/
    M5.begin();             // Set up the board
    //…
    /*** Set up the M5.Lcd ***/
    M5.Lcd.fillScreen(BLACK);
    //...
    /*** Set up the scale ***/
    scale.begin(&Wire, 32, 33, DEVICE_DEFAULT_ADDR, 100000U);
    //...
    /*** Set up wifi ***/
    transport.setWifiSsid(WIFI_SSID);
    transport.setWifiPass(WIFI_PASSWORD);
    //...
    /*** Set up the Prometheus client ***/
    client.setUrl(GC_URL);
    client.setPath((char*)GC_PATH);
    client.setPort(GC_PORT);
    //...
    /*** Add the timeseries to the write request ***/
    req.addTimeSeries(ts_weight);
}
```

Water Sensor

```
void loop() {
    int64_t time = transport.getTimeMillis();
    float weight = scale.getWeight();

    /*** Show the measurement on the display ***/
    M5.Lcd.printf("%.1fg", weight);
    if (weight < WATER_ONE_THIRD + BOWL_WEIGHT)
        M5.Lcd.printf("Empty!");
    else if (weight < WATER_TWO_THIRDS + BOWL_WEIGHT)
        M5.Lcd.printf("Good");
    else
        M5.Lcd.printf("Full!");

    /*** Add the measurement ***/
    ts_weight.addSample(time, weight);

    /*** Send the measurement ***/
    PromClient::SendResult res = client.send(req);
    if (ts_weight.addSample(time, weight))
        ts_weight.resetSamples();

    /*** Wait 5 seconds before taking next measurement ***/
    delay(5000);
}
```

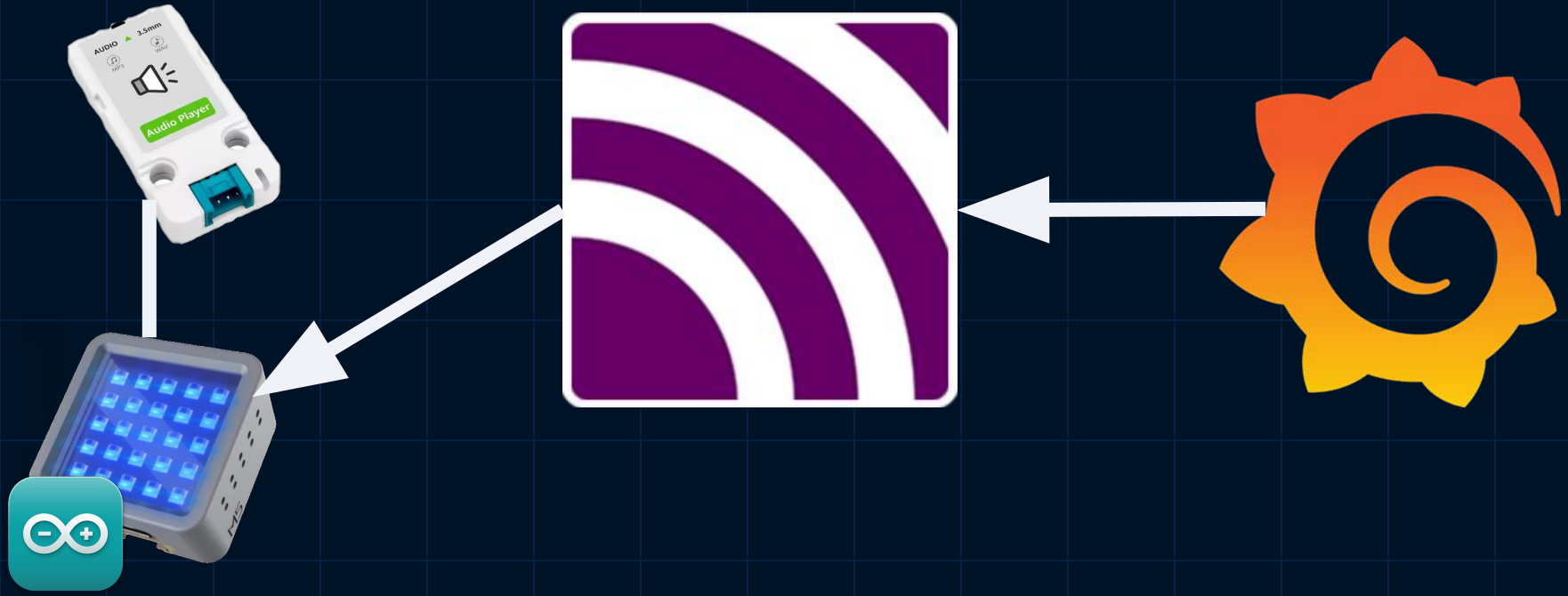
DEMO

MQTT



All messages have a “topic” that nest in a path like `dog/alerts`

Alerting System



Alerting System

```
void mqttReceived(String &topic, String &payload) {

    /*** Check to see if the message we got was the topic we're subscribed to ***/
    if (topic == MQTT_TOPIC) {
        // We could / should actually parse the JSON object we get from the alert.
        // But instead, we'll just look to see if it contains "status"...
        if (payload.indexOf("status") != -1) {
            // ... and if so, if that status contains "firing".
            // This is _not_ super robust, but it's for a fast demo.
            if (payload.indexOf("firing") != -1)
                alerting = true;
            else
                alerting = false;
        }

    }
}
```

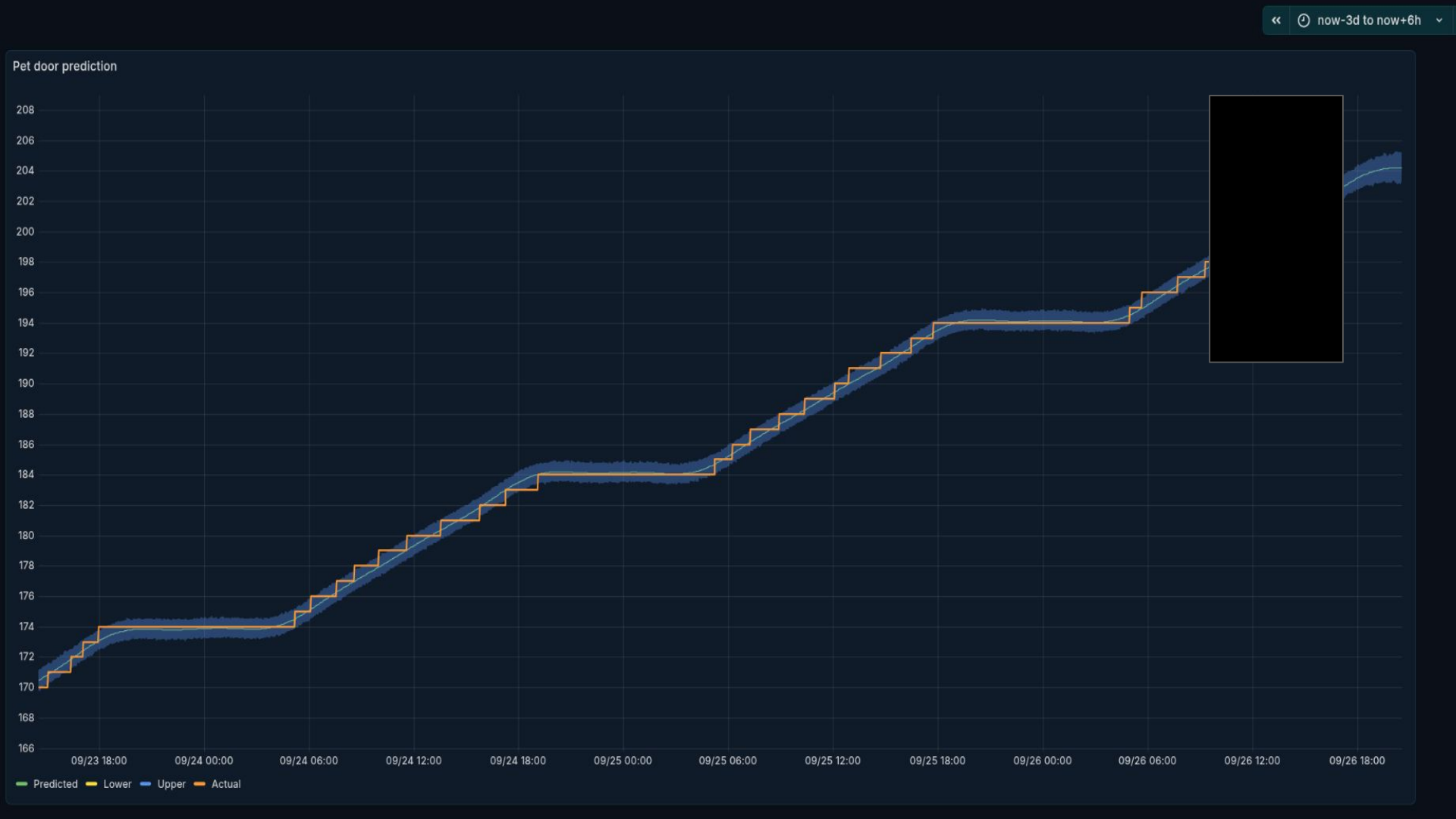
Alerting System

```
void loop() {
    static bool prev_alert_state = false;

    /*** See if there's an MQTT message for us ***/
    mqtt.loop();

    if (alerting) {
        draw_face(leds, FACE_SAD);
        if (!prev_alert_state)
            audioplayer.playAudio();
        prev_alert_state = true;
    }
    else {
        draw_face(leds, FACE_HAPPY);
        FastLED.show();
        audioplayer.stopAudio();
        prev_alert_state = false;
    }
}
```

DEMO

Data analytics

https://rmtesting.grafana.net/d/rmskfjz/pet-door


now-3d to now+6h
Pet door prediction
208
206
204
202
200
198
196
194
192
190
188
186
184
182
180
178
176
174
172
170
168
166
09/23 18:00
09/24 00:00
09/24 06:00
09/24 12:00
09/24 18:00
09/25 00:00
09/25 06:00
09/25 12:00
09/25 18:00
09/26 00:00
09/26 06:00
09/26 12:00
09/26 18:00
Predicted
Lower
Upper
Actual

Thank You!

ronald@grafana.com

github.com/mccollam