

# ProximaSafe: Joining the Dots in OCI to build a Stream Analysis Lab

---

## Chapter Three: Sensors, Pipelines and back to Edge

*Wandering the face of the Earth*

*Wondering what our dreams might be worth*

*Learning that we're only immortal*

*For a limited time*

(Neil Peart, 1991)

## What we've done so far

*Greetings*, and welcome to **Chapter Three**, the last of this series. In the *previous articles* we set up all the bits and pieces we need to build a portable lab aimed to study and develop stream analysis, such as:

- All the **Oracle Cloud Infrastructure** artifacts needed, such as Golden Gate Stream Analytics, OCI Compute running the MQTT service, Oracle Functions, API management and the wiring to connect the services.
- **Edge** components mostly made on Raspberry Pi, ESP32 elements and Arduino, easy to procure, setup and stow away

Now it's time to design some example use cases that imply:

- Use Cases design & considerations
- Assign a task to each of the edge components
- Design and implementing pipelines within Stream Analytics using patterns included with the platform

## Use Cases

---

We can think of - at least - three scenarios using the technologies set up during the previous chapters and effectively make the most of stream analysis:

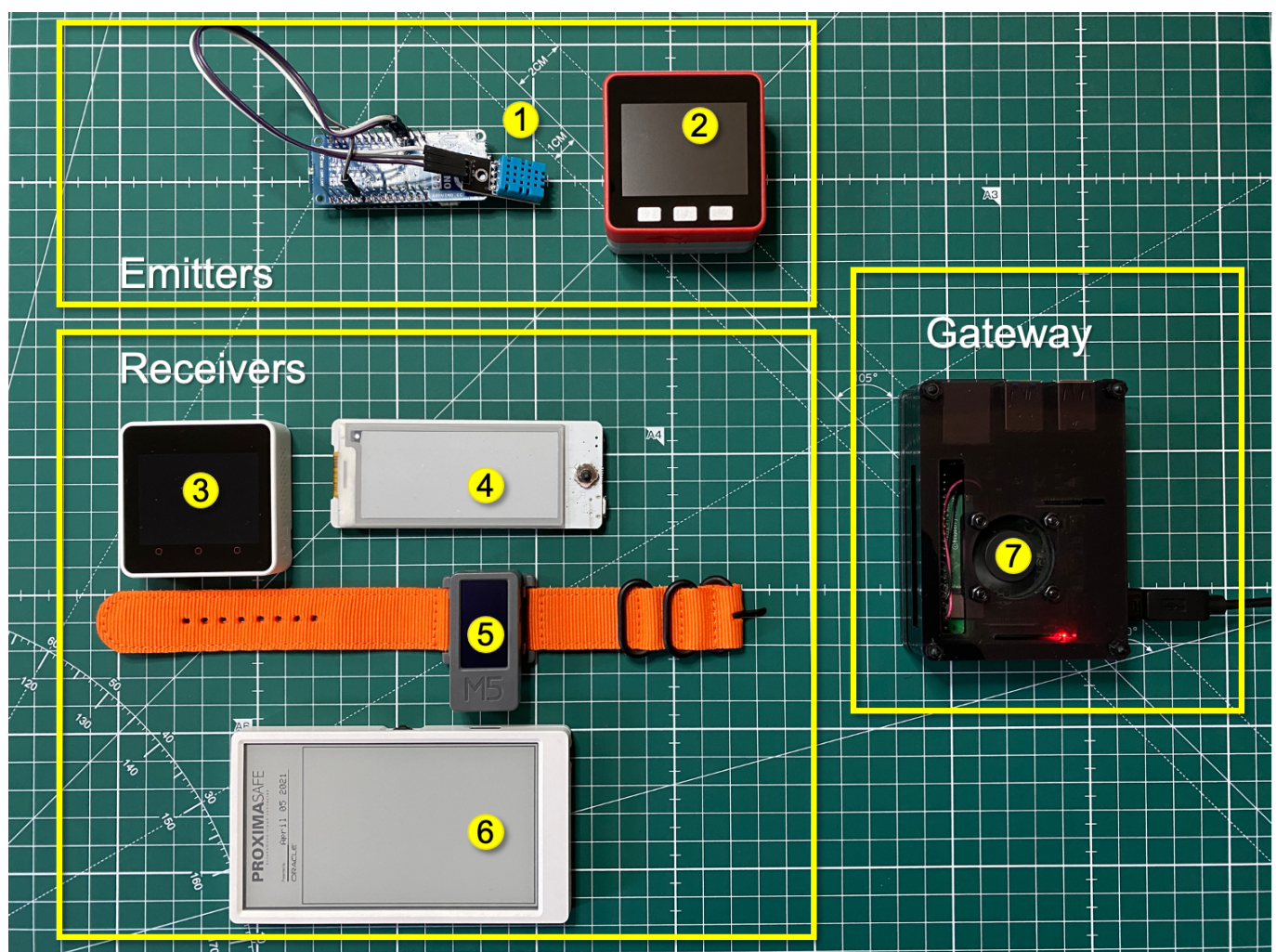
- Checking if the **environmental parameters** of a given perimeter (a closed space, i.e. an university classroom or an office) are *safe*: if over a threshold or if there's an up trend in temperature and/or humidity, the system will be required to send an alarm back to the edge to take appropriate actions.
- Checking if the **number of events** in a determined point and within a certain number of seconds indicates an abnormal condition: for instance, a sensor in a one-way corridor that simply count the passage of people. Should the parameters go over a threshold within a number of seconds or if an *up trend* is detected, we could decide to stop the people accessing the corridor by turning on a red 'traffic light' to mitigate the chance of a gathering.
- Check if **sanitizers** in specific locations are malfunctioning (empty gel), or are being tampered with by checking messages sent by the dispensers: if the stream analysis module receives two alarms

within a time range (few seconds) the we'll warn the person responsibly of maintenance by sending a message to a wearable smart tag and solve the problem.

These simplistic use cases do not fully take into account all the possibilities offered by selecting a **Geofence** in Golden Gate Stream Analytics, that is a cool and useful function for scanning what is happening within a defined set of coordinates, but could serve as basis for more sophisticated (and real!) analysis performed on any size of environment. Nevertheless, we'll hardwire each flow to different coordinates mocking the presence of sensors in the real world.

## Assign the tasks to Edge components

Here's the map of the components we're going to use (and program):

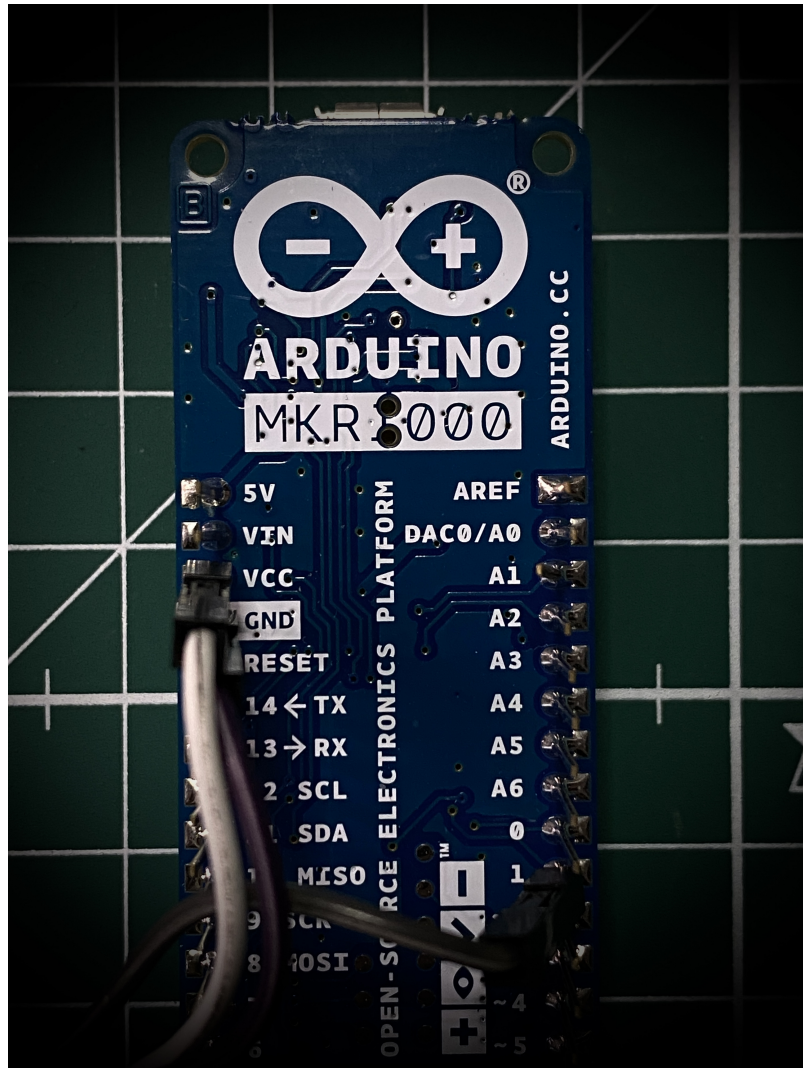


Please find the code for each of these boards at this address (NOTE: insert the GitHub link, open in a new window/tab).

### 1 - Arduino with DHT11 sensor (Environment)

This configuration is the 'Hello World' equivalent of the Arduino programming. The board is an **Arduino MKR1000**, sporting a 32-bit, low power ARM MCU provided by the Atmel ATSAMW25 SoC. Wi-Fi and MQTT features are provided by the [WiFi101.h](#) and [pubsub.h](#) libraries, respectively. No GUI is provided,

except for the Serial Monitor available in Arduino IDE, so we're (possibly) going to use the `Serial.println()` extensively.



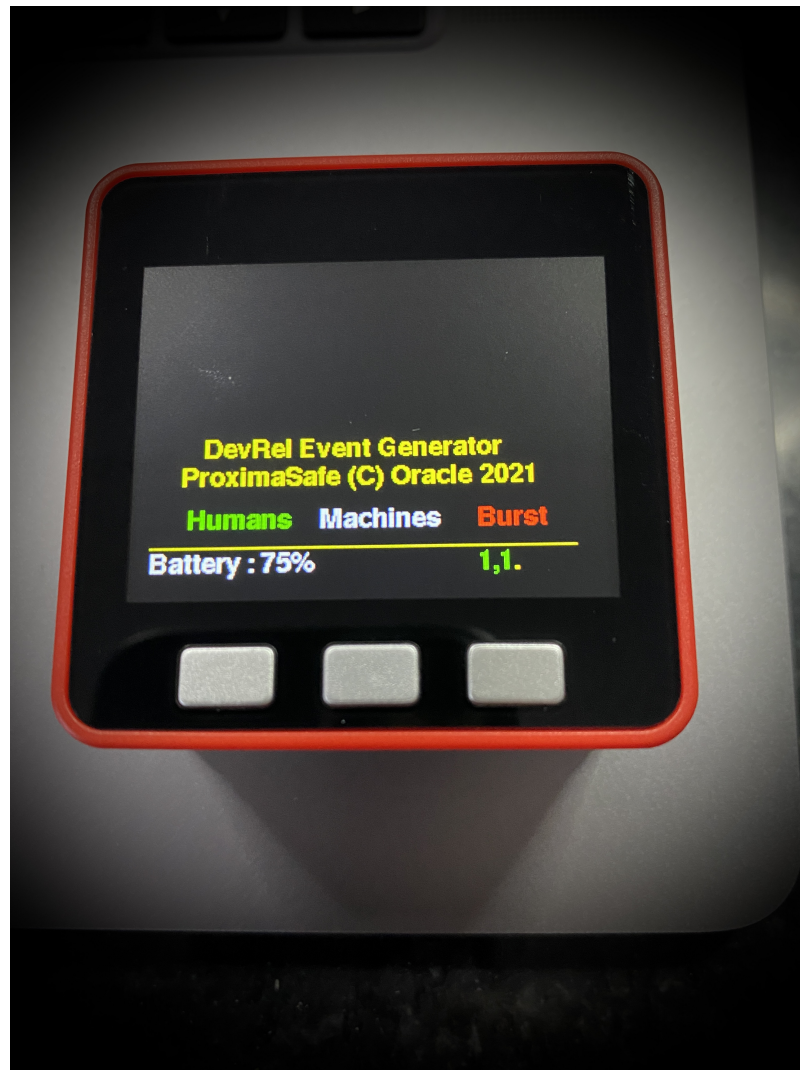
As I mentioned in one of the previous chapters, I tried to switch to a more compact sensor (a BMP280 *hat* connected to a M5StickC) in order to avoid a scruffy jumper-based connection between the MKR1000 and the DHT11, but the temperature and humidity measured is heavily influenced by the heat generated by the M5, which kinda misses the objective. So, we'll remain with the MKR1000 - and, believe me or not, it's definitely not a leftover from the Proxima City days, I just love the board! This simple board will send information about temperature and humidity that'll be intercepted by Stream Analytics to verify up/down trends and eventually go downstream with possible anomalies.

## 2 - M5Stack Fire (Event Generator)

The M5Stack Fire (a.k.a the *Red One*) is based on the M5 packaging and it's based on the **ESP32** architecture. This box is the primary event generator, which downsizes - a bit - its capabilities (it's got even an IMU posture sensor, that would be a nice feature to enhance the tampering use case), so it'll fire events simulating the people flowing in a corridor and events from sanitizers.

Fortunately, most of the M5 have a tiny cute display, so we'll be able to expose feedback when sending/receiving messages to OCI.





The UI I've programmed is *really* basic (in fact, it's character based), with the addition of battery level detection (found the API call somewhere, and it's not accurate nor reliable, as I discovered), the number of messages sent, and results. The buttons are programmed to send a single message or burst of messages to trigger a pattern-based analysis in Stream Analytics.

### 3 - M5Stack Core 2 (Alarm Detector)

The M5Stack Core 2 (a.k.a the *White One*) is an evolution from the previous M5 box, it's got touch sensors and another bunch of goodies that we won't use, since the its main task is to display alarms for corridors and possibly other locations where we need to show an alert - in a form of a red traffic light.





This box could be expanded with an external display via the Grove port or a relay as an actuator (actually, I've put some code for the external LCD RGB display but I'm not using that).

## 4 - E-ink Smart Badge (Badgy)

The **Badgy** device is equipped with an e-ink display and the **ESP8266** chip. While e-ink refresh rates aren't ideal for contents changing rapidly, it is certainly good for the *Sanitizers* use case to report to the maintenance employee which dispenser machine has incurred into troubles, in order to fix it quickly even when on the move during regular maintenance routines.



## 5 - M5Stick Wearable Beeper

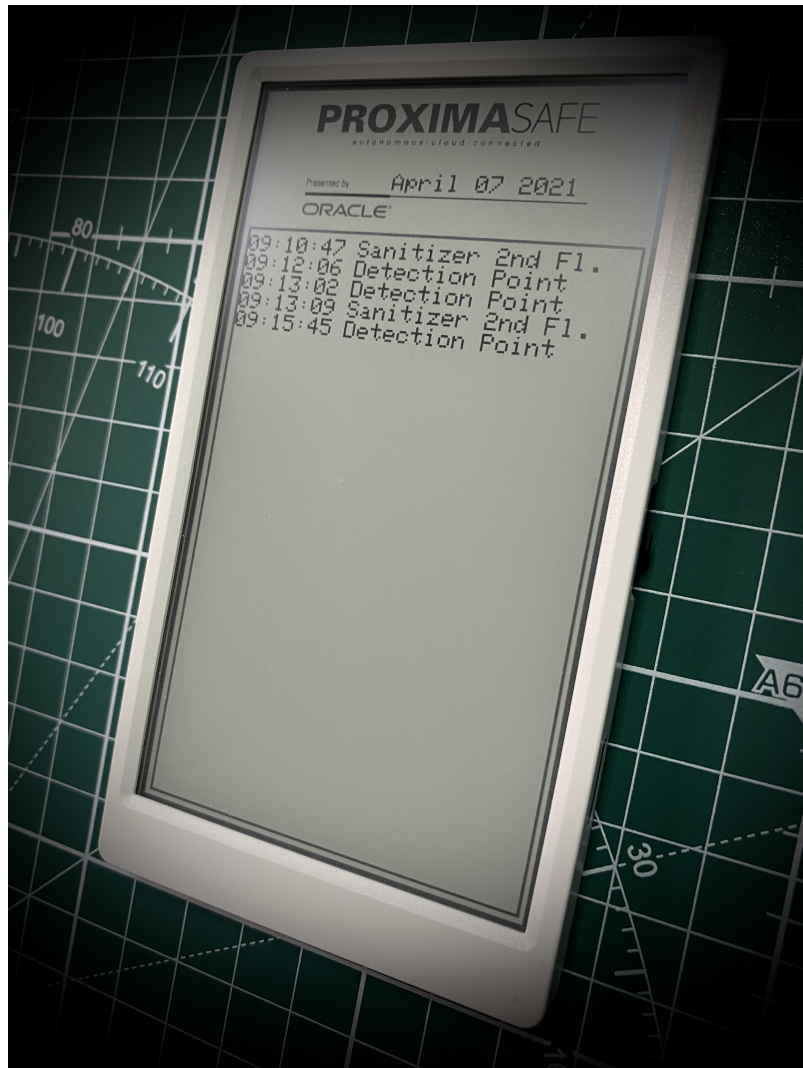
In alternative to the Smart Badge, the gray M5Stick can be worn as a watch/beeper receiving alarm messages from OCI. The M5Stick (a.k.a the *Grey One*) is even smaller than the previous M5, it's still based on the ESP32 architecture and it sports Wi-Fi, Bluetooth and a IR blaster.



## 6 - M5Paper Billboard

The **M5Paper** is the last addition to the M5 family: e-ink, capacitive touch screen, a beefy battery and a wide display - looks like we just found the billboard for our microlab! This device will be in charge to display, one page at a time, all the alarm events coming from OCI with date and time.





Since it's a full-blown elnk minitablet with touch support, I suppose I should implement a nicer UI touch-based, to allow the drill down of a particular message to get more info about it. This could be an idea for a further development.

## 7 - Raspberry Pi 4B as the Gateway

We've configured, during the previous chapters, the **Raspberry Pi** to act as our local MQTT server (Mosquitto-Edge) bridged to the MQTT server resident on a OCI Compute instance (Mosquitto-Cloud), and now it's time - at last - to let the other boards/sensors send message to this thingie and see the loopy conversation Edge » Cloud » Edge somewhat effective.



Most of the M-Fivers share the same code snippets:

- Getting a **Wi-Fi connection**

```
WiFi.begin(ssid, pass);

// Wait for connection
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}

Serial.println("");
Serial.print("Connected to ");
Serial.print(ssid);
Serial.println("");
Serial.print("IP address: ");
Serial.print(WiFi.localIP());
Serial.println("");
```

- Connect to the **MQTT server** running in the Raspberry Pi

```
mqttClient.setServer(BROKER,MQTT_PORT);
mqttClient.connect(clientId);

Serial.println("Connecting to MQTT Broker");
while (!mqttClient.connected()) {
  if (mqttClient.connect(clientId)) {
    Serial.println("Waiting for MQTT Broker");
    Serial.print(".");
    delay(500);
  }
}
```

- Getting time from an NTP server and print it in a human-readable form

```
// Get time considering Time Zone and Daylight Offset
configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
struct tm timeinfo;

if (!getLocalTime(&timeinfo)) {
  Serial.println("No Time from NTP Server");
}

strftime (buf, sizeof(buf), "%B %d %Y", &timeinfo);
```

- Subscribing to a topic: in this case we'll subscribe to the **cloud/alarm** topic

```
mqttClient.subscribe(TOPIC);
Serial.println("Subscribed to MQTT topic");
```

and process the received messages with a callback, showing the alerts in different colors

```
void DisplayCallback(char* topic, byte* payload, unsigned int len)
{
  Serial.println((String)topic);
  Serial.println(len+1);

  if ((String)topic == ALARM_TOPIC) {
```



```

// Save Message
char msg[300];
for (int i=0; i < len; i++)
    msg[i] = payload[i];
msg[len] = NULL;

// Issue an audio alarm
AudioEffect();

// Filter message Status
if ((String)msg == "{\\\"STATUS\\\":\\\"Detection Point\\\"}") {
    M5.Lcd.fillScreen(TFT_RED);
    M5.Lcd.setCursor(10, 40);
    M5.Lcd.setTextColor(TFT_BLACK,TFT_RED);
    M5.Lcd.setTextSize(2);
    M5.Lcd.println("Gates Alert");
}
if ((String)msg == "{\\\"STATUS\\\":\\\"Sanitizer 2nd Floor\\\"}") {
    M5.Lcd.fillScreen(TFT_ORANGE);
    M5.Lcd.setCursor(10, 40);
    M5.Lcd.setTextColor(TFT_BLACK,TFT_ORANGE);
    M5.Lcd.setTextSize(2);
    M5.Lcd.println("Sanitizer Alert");
}
if ((String)msg == "{\\\"STATUS\\\":\\\"DHT11\\\"}") {
    M5.Lcd.fillScreen(TFT_ORANGE);
    M5.Lcd.setCursor(10, 40);
    M5.Lcd.setTextColor(TFT_BLACK,TFT_ORANGE);
    M5.Lcd.setTextSize(2);
    M5.Lcd.println("Environment");
}
} else if ((String)topic == ALARM_CLEAR) {
    M5.Lcd.fillScreen(TFT_GREEN);
    M5.Lcd.setCursor(10, 40);
    M5.Lcd.setTextColor(TFT_BLACK,TFT_GREEN);
    M5.Lcd.setTextSize(2);
    M5.Lcd.println("Ready");
}

delay(100);
}

```

- Publishing a message from the Event Generator by pressing a button

```

if (M5.BtnA.isPressed()) {
    M5.Lcd.clear(BLACK);
    M5.Lcd.setCursor(10, 40);
    M5.Lcd.println("Sending Message...");

    char* msg = formatPeopleMessage(msgbuf, "Gates", "Detection Point",
    timestamp, "People passing", fLatGate, fLonGate);
}

```

```

    Result = mqttClient.publish(PEOPLE_TOPIC, msg, true);
    M5.Lcd.setCursor(10, 60);
    if (Result)
        M5.Lcd.println("Sent.");
    else
        M5.Lcd.println("Not sent.");

    M5.Lcd.setCursor(10, 220);
    M5.Lcd.setTextColor(TFT_WHITE);
    M5.Lcd.printf("Battery : %i%%", getBatteryLevel());
    ledFeedback(PEOPLE_EVENT);
    delay(100);
}

```

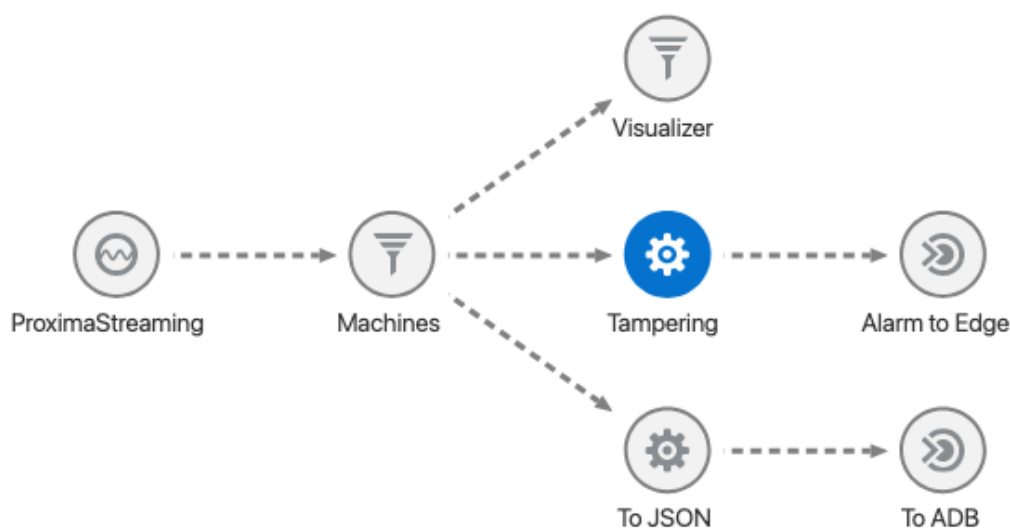
Please consider this code just as a reference: most of the times I struggle to remember what I did during the previous tinkering sessions, so I'm all in for code that speaks for itself.

## Pipelines setup

---

### Check tampering with Sanitizers

Here's the first pipeline, using the interactive designer included with Stream Analytics.



Translating the visual pipeline into words, here's what it does:

- Accept messages from OCI Streaming as a source, defined in the Catalog
- Query all the messages where the field `sensors` contains the text `Sanitizers`. Then, three actions take place:

- Produce a geospatial visualization based on *Latitude* and *Longitude* coordinates located in the message.
- Check if the message from the same sanitizer has been sent previously within a short time span via the pattern '**A** followed by **B**'. If this pattern is detected, we'll invoke the OCI Function previously created in Chapter Two to send the alarm back to edge.
- Format the message to JSON via a **To Json** pattern and store the message in an Autonomous DB defined as a target in the Catalog (I think that could be a nice addition).

Any alarm should result in a message sent to the OCI Function `mqtt_pub` which in turn routes the message to the Mosquitto-Cloud instance, and eventually sent to the edge via MQTT Bridging, using the `cloud/alarm` topic.

The message will be shown on the **Badgy** and on the **Billboard** as well, since these devices are listening on this specific topic:

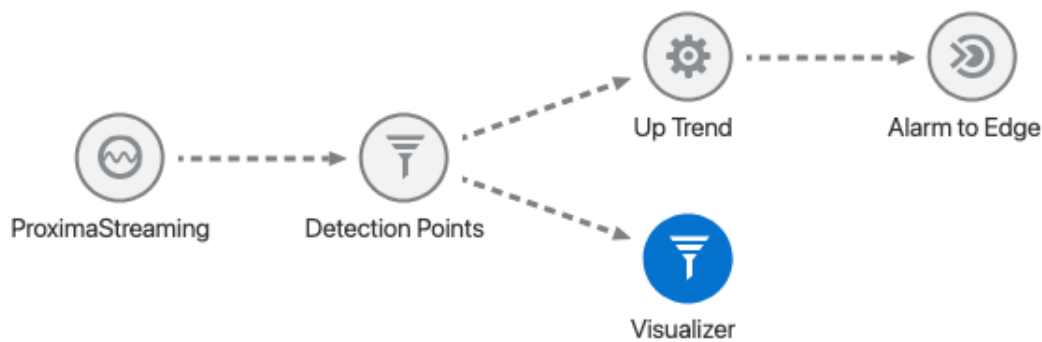


The **Beeper** can also be triggered with an audio alert. Should you be interested in having the audio file played on the alert event, I cannot publish that. It's copyrighted (*Jimi Hendrix's "Purple Haze"*).

## Detection Points & Environment alarms

Here's the detection points pipeline, which can be triggered by push the **Burst** button on the Event Generator (the *Red* box).





Stunningly similar to the previous one, here are the inner workings:

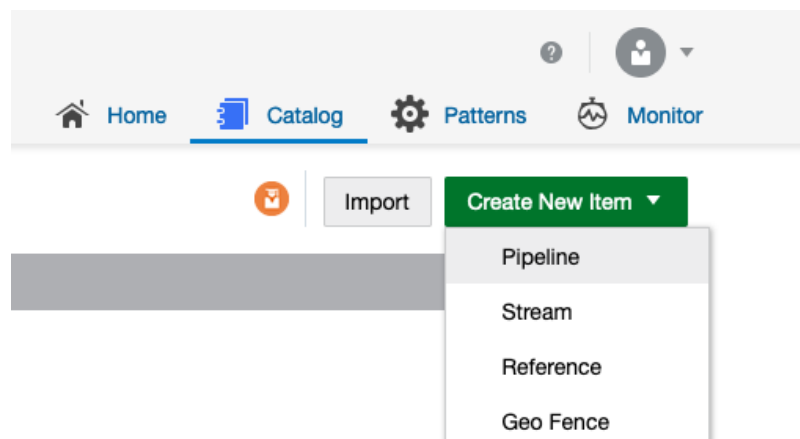
- Accept messages from OCI Streaming as a source, defined in the Catalog (as before).
- Query all the messages where the field **sensors** contains the text **Detection Point**. Then the actions:
  - Produce a geospatial visualization based on *Latitude* and *Longitude* coordinates located in the message (as before).
  - Check if messages from the same detection point have been sent previously within a time span (30 seconds, maybe?) via the pattern **Up Trend**. If true, we invoke the OCI Function previously created to send the alarm back to edge.

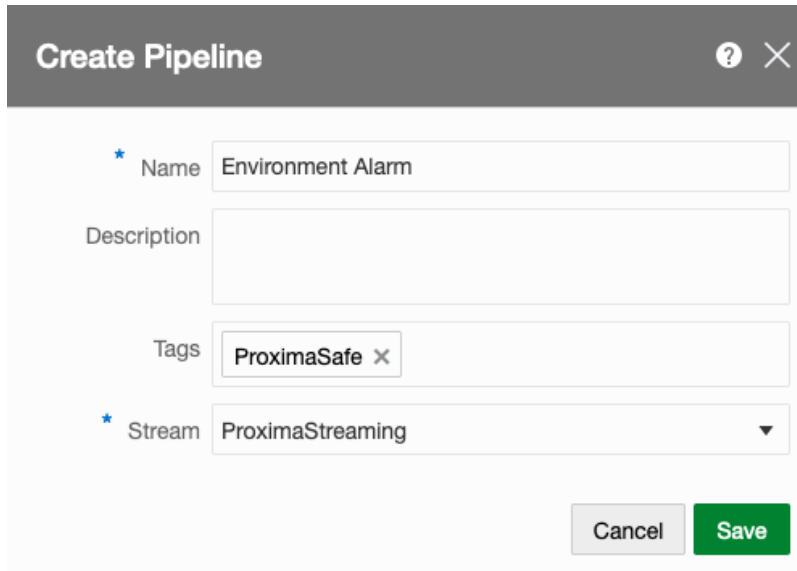
In this case we should light up a *red traffic light* where the detection point is located in order to discourage (or deny) the access to that gate, corridor, or location. Since all of the edge components are miniaturized, we'll just display a red alert within our Alarm Detector M5 box:



The Environment alarm pipeline could be designed with the same steps, analyzing the temperature and humidity changes in order to detect an up trend, and clear the alarm when environmental values are within an acceptable range. Let's follow the pipeline construction steps in details:

- From the Catalog Menu, let's create a new Pipeline selecting our Stream.





**Create Pipeline** ? X

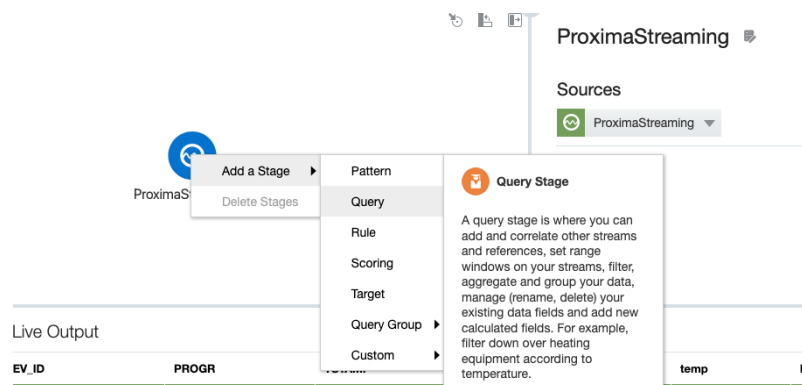
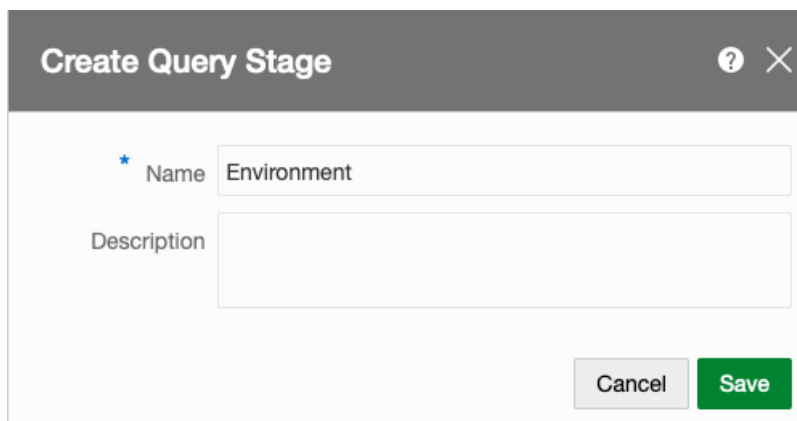
★ Name

Description

Tags

★ Stream

- A basic canvas should be displayed with the name of the Stream involved. Then, after a right click on the Stream icon, we are presented with a menu to add a Stage: selecting the Query stage, we'll make sure to process only the environment data coming from the DHT11 sensor.

**Create Query Stage** ? X

★ Name

Description

- This query on the *data in motion* will select messages coming from our environmental sensor by selecting the *Filters* tab in the upper right menu and inserting some simple query data, such as:



## Environment

 Detach

Sources **Filters** Summaries Visualizations

☒ Match All ☐ Match Any

sensor ▼

contains ▼

DHT11






Text 

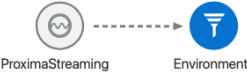



Add a Condition

Add a Group

- It's a simple and intuitive mechanism can be applied anywhere in Golden Gate Stream Analytics, that makes the construction of pipeline easy and intuitive! Even without committing the pipeline in production mode, you can see immediately the effects in real time: note that on the **Live Output** pane only the DHT11 events are displayed.






← Done Environment Alarm 
 Draft | 
  Publish | 
 





Environment 

Sources **Filters** Summaries Visualizations

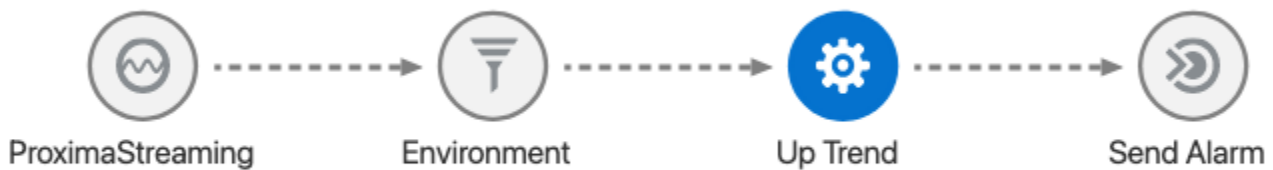
sensor contains DHT11

Environment - Live Output 



 Columns 
  Detach

sensor	DEV_ID	PROGR	TSTAMP	STATUS	temp	hum	Lon	Lat
DHT11	Environment		272	1617869928	Smart Env OK			
DHT11	Environment		271	1617869923	Smart Env OK			
DHT11	Environment		270	1617869918	Smart Env OK			
DHT11	Environment		269	1617869913	Smart Env OK			
DHT11	Environment		268	1617869908	Smart Env OK			
DHT11	Environment		267	1617869903	Smart Env OK			
DHT11	Environment		266	1617869898	Smart Env OK			
DHT11	Environment		265	1617869893	Smart Env OK			
DHT11	Environment		264	1617869888	Smart Env OK			
DHT11	Environment		263	1617869883	Smart Env OK			
DHT11	Environment		262	1617869878	Smart Env OK			
DHT11	Environment		261	1617869873	Smart Env OK			

ProximaStreaming

- We can proceed with pipeline build by adding an up trend pattern selecting temperature or humidity as the **Tracking Value** and insert a call to the OCI Function and will return an alarm to the edge if the conditions are met:



Every time that we feel that the pipeline is ready (and we can test any part of it while designing it, interactively), we can commit the pipeline (note that the target elements are activated only the pipeline is in the **Published** state).

## Epilogue

---

Setting up a personal development laboratory with the help of few development boards, sensors and some OCI resources has never been easier (and fun!). Every test has been made connecting the edge elements to a wireless router 4G-enabled that I acquired some time ago (well, I don't have a full 4G connection where I live, it's a frontier signal zone), so the connection to Oracle Cloud Infrastructure is just another variable factor in evaluating the lab performance, which - anyway - is frankly good.

Once the **OCI function** has been warmed up (by means of test messages sent at irregular intervals) the round-trip time between the anomaly phenomenon induced in the edge lab and the alarm returned and processed in the edge is almost **instantaneous**. To shed a light regarding the things to improve on this base configuration, let's examine the following considerations:

- Being a development lab based on development boards, everything can be optimized (my code for sure).
- While the local Wi-Fi used by the boards is steady, the 4G (or 3G, most of the times) available connection used to reach the OCI resources from this microlab represents a 'real world' connectivity example (no 5G here in the neighbourhood...).
- The MQTT to OCI Python router could be containerized and made more robust within an orchestrator ([OKE](#)).
- We could use OCI compute to setup a number of VMs injecting load to the Cloud components via [siege](#), [vegeta](#) or any burden-related generators to observe the behaviour of the system.
- Any other thing that is fun

I believe that we could reuse the concept, maybe with different sensors, for a variety of use cases that encompass the usage of actual industrial sensors, geofences and other goodies.

Thanks for your patience, and see you on Zoom.

Zip and Zest!