Paper 6015-2020

Unlock the Business Value of IoT with Analytics

Jon Klopfer, Pinnacle Solutions, Inc.
Elizabeth McGlone, Pinnacle Solutions, Inc.
Donald L. Penix, Jr. (DJ), Pinnacle Solutions, Inc.

ABSTRACT

The impact of Internet of Things (IoT) Analytics might be greater—and taking hold more quickly—than many observers expect. Organizations that are pursuing an IoT strategy are finding they can't compete effectively without using analytics. Despite the importance of IoT Analytics, many organizations lack a clear vision to execute their initiatives. According to a recent study by PWC, "more than six out of ten organizations have failed to take operational IoT initiatives past proof-of-concept stage or beyond implementation." This paper provides an example IoT setup using SAS® software, open source software, and a Raspberry Pi® to build a project that can be utilized for demonstrating IoT Analytics. Why would we do this? Although not covered in detail in this paper, we can use IoT Analytics to help with the following use cases:

- Predictive maintenance
- Image recognition and analysis
- Process optimization
- Overall equipment effectiveness
- Refinery monitoring
- Risk analysis
- Operational performance
- And more

It's time to make your IoT data work for you!

INTRODUCTION

SAS Event Stream Processing (SAS ESP) is a powerful real-time analytics engine, but we can extend the capabilities with some custom dashboard development. Customers using SAS Analytics for IoT need a flexible, lightweight web interface with customizable dashboards that will give them the best of both worlds: the power of SAS ESP and visualizations of the data and insights that matter the most. Using open source tools, we can create completely custom dashboards that deliver on these needs and are built using widgets, enabling easy reproducibility and customization for project implementations. This solution is an illustration of how combining the SAS Platform, a simple IoT sensor device, and open source software gives the customer a preliminary dive into IoT Analytics.

This solution was developed in parallel with a demo using a Raspberry Pi designed to give a quick example on how to present the power of SAS ESP and real-time analytics. The specs for the Raspberry Pi "ShakeBox" demo are included for reference. All aspects described in the Technical Scope section, including the ShakeBox hardware and software, are part of the solution that is deployed in the example environment.

This paper's purpose is to outline our innovative solution that combines the power of the SAS platform, a simple IoT device, and open source tools to deliver greater value and unlock the business value of IoT analytics. Within the following, you will learn how to enhance SAS ESP in five main ways:

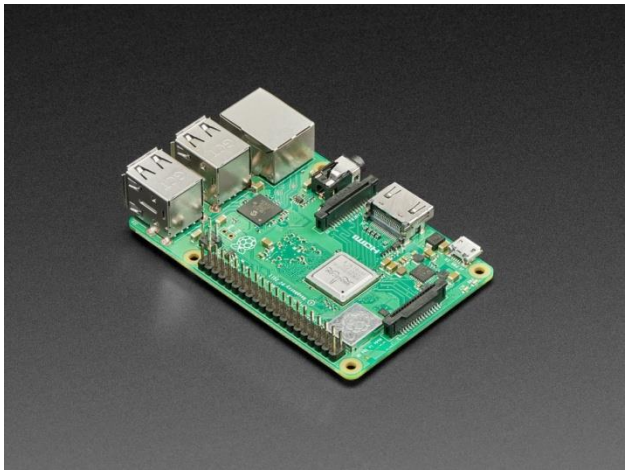
1. Data consumption/processing
2. Data visualization
3. Software stability
4. Data quality management
5. Server stability

TECHNICAL SCOPE

This paper will break down the technical scope of the solution into eight sections:

1. ShakeBox hardware (Raspberry Pi)
2. ShakeBox software
3. Flat file processor
4. CnC command line utility
5. Logic engine
6. RabbitMQ hardware and software
7. SAS ESP hardware and software
8. Serverless web interface

SHAKEBOX HARDWARE COMPONENTS



Display 1. Raspberry Pi 3 B+

We utilized the following hardware components to build the IoT "sensor" device.

- Raspberry Pi 3 B+
- Sense HAT
- Battery
- Case
- Cables
- SD Card
- Wall Charger

The Sense HAT was utilized due to the various data collection components that are typical for an IoT implementation. Those include a gyroscope, an accelerometer, magnetometer, barometer, temperature, and relative humidity.

The specifications are as follows:

Sense HAT Sensors:

- Gyroscope - Angular rate sensor (dps): ~245/500/2000
- Accelerometer - Linear acceleration (g): ~2/4/8/16
- Magnetometer - Gauss: ~4/8/12/16
- Barometer: 260 – 1260 hPa absolute range
- Temperature: - 0-65°C range
- Relative humidity: 20-80%rH range

SHAKEBOX SOFTWARE

- Raspbian operating system (Debian flavor of Linux specifically modified to work well with Raspberry Pi hardware)
- Python 3.6
- Python-virtualenv

The following source code is an example configuration file for the ShakeBoxes:

```
# SHLEX FORMATTED CONFIG FILE FOR SHAKE BOXES
# TAB DELIMLIEMTLEIEMD -- TAB!

# Config for shakebox --
[RMQSensorConfig]
    #RabbitMQ Config
    host=[redacted]
    user=[redacted]
    passwd=[redacted]
    hwid=shakebox1
    # RabbitMQConnection Additional Config
    exchange=[redacted]
    queue=[redacted]
    queuegps=[redacted]
    auto_connect=True
    verbosity=11

[FeedbackLoop]
    #RMQ config for the feed back loop
    host=[redacted]
    user=[redacted]
    passwd=[redacted]
    # JSON top level target=hwid is you- grab from RMQSensorConfig
    # target=global you all follow in line
    # Fannout exchange
    exchange=feedbackfanout
    # cnc queue just for sure
    queue=cnc
    auto_connect=True
    verbosity=11
```

The following source code is an example file for the launching the ShakeBoxes:

```
import sys
from util.NojDispatcher import NojDispatcher

from SensorRecord import SensorRecord
from LEDDisplay import LEDDisplay
from FeedbackLoop import FeedbackLoop
from GPSSensor import GPSSensor
```

```
import time
import urllib3
from urllib.request import urlopen

def internet_on():
        try:
                response = urlopen('https://www.thepinnaclesolutions.com/',
timeout=10)
                return True
        except:
                return False

pinnacle_orange = (218,109,29)

if __name__ == '__main__':
        # get recorder ready and display
        display = LEDDisplay()
        feedbackloop = FeedbackLoop(display=display)
        #Create dispatcher
        dispatcher = NojDispatcher()
        display.clear()
        attempts=0
        dispatcher.blockingTask(display.text, "Checking for Internet!",
forground=pinnacle_orange, speed=.04)
        while not internet_on():
                attempts+=1
                dispatcher.blockingTask(display.text, "Failed: Internet NOT
FOUND. WiFi MAY NOT BE READY. Trying again - ATTEMPT # "+str(attempts)+"...
", speed=.04)
                time.sleep(1)
        dispatcher.blockingTask(display.text, "Connecting RMQ",
forground=pinnacle_orange, speed=.02)
        dispatcher.blockingTask(feedbackloop.connectRMQ)
        dispatcher.blockingTask(display.text, "Booting GPS",
forground=pinnacle_orange, speed=.02)
        gpsrecorder = GPSSensor("RMQSensorConfig",display)
        dispatcher.blockingTask(gpsrecorder.connectRMQ)
        dispatcher.blockingTask(display.text, "Booting SensorHat",
forground=pinnacle_orange, speed=.02)
        try:
                recorder = SensorRecord("RMQSensorConfig", display)
                dispatcher.blockingTask(recorder.connectRMQ)
        except:
                dispatcher.blockingTask(display.text, "FAIL!")
                for x in range(5):
                        display.backgroundPulse((254,0,0), pinnacle_orange,
steps=8, delay=.02)
                        display.backgroundPulse(pinnacle_orange, (254,0,0),
steps=8, delay=.02)
                        display.text("Failed2Connect! RESTART ME! "+str(5-x),
forground=display.getRandomColor(), background=None, speed=.05)
                display.clear()
                dispatcher.blockingTask(display.text, "X")
                raise
                sys.exit()
        dispatcher.blockingTask(display.text, "Success!",
forground=pinnacle_orange, speed=.02)
        for x in range(5):
```

```
                display.backgroundPulse((0,0,0), pinnacle_orange, steps=8,
delay=.02)
                display.backgroundPulse(pinnacle_orange, (0,0,0), steps=8,
delay=.02)
                display.text(str(5-x), forground=display.getRandomColor(),
background=None, speed=.1)

        recorder_thread = dispatcher.asyncTask(recorder.start)
        display_thread = dispatcher.asyncTask(display.start)
        feedbackloop = dispatcher.asyncTask(feedbackloop.start)
        gpsrecorder_thread = dispatcher.asyncTask(gpsrecorder.start)

        try:
                while True:
                        time.sleep(.01)
        except KeyboardInterrupt:
                dispatcher.blockingTask(display.text, "KILLING Recorder")
                display.clear()
                recorder.kill()
                recorder_thread.join()
                time.sleep(1)
                dispatcher.blockingTask(display.text, "Killing GPS")
                gpsrecorder.kill()
                gpsrecorder_thread.join(30)
                display.kill()
                display_thread.join()
```

FLAT FILE PROCESSOR

The flat file processor – we built with open source tools - breaks down comma separated value (CSV) files and other formats into flat files so that the data can be fed through RabbitMQ into SAS ESP for analysis. It monitors the directory for the creation of files and turns files into single records so they can be processed as events in SAS ESP. One program inserts file locations into a queue. Another program reads from the queue, converts into single records, and sends the "events" to RabbitMQ and then deletes the file. RabbitMQ is then read into SAS ESP. If anything in the chain of events goes down, the flat file processor can toss its backup data back through, allowing for failover.

On its own, SAS ESP can process CSV files as dimensional tables or as transactional stores, but not as events. Without a system like our flat file processor, the only other option for developers is to develop middleware that alerts SAS ESP when an update is being made. Developing this middleware on top of an existing system of software and applications can be tricky if you have to update it every time there is an update in your ERP, CRM, or other technology. The growing CSV file feature built into SAS ESP is not designed for this sort of kill and refill data dumps. The flat file processor works seamlessly with a wide variety of existing systems and it will always work regardless of changes to the system or technology stack. This gives you the flexibility to add and modify the system without fear of breaking the ability to read CSV files into SAS ESP as events.

This processor is a noninvasive way for you to use CSV files for data collection and leverage the real-time analytics power of SAS ESP. The flat file processor provides efficient buffering, with the ability to process and send RabbitMQ a benchmark average event rate on a low-end machine of around 4000 records/seconds per process. The following source code is an example configuration file for this flat file processor:

```
#Python watchdog directory watcher, file processor and special rules for
CSV parser
[FileWatcher]
        my_id=watcher_worker
        incoming_directory=/sas/flatfiles/processing

        #RabbitMQ Config
        host=[redacted]
        user=[redacted]
        passwd=[redacted]
        #RabbitMQConnection Additional Config
        exchange=[redacted]
        queue=pending_files
        auto_connect=True
        rmq_logger=DirectoryListenerLogger
        verbosity=11

[NetworkFileMover]
        target_incoming_directory=/sas/flatfiles/incoming

[DirectoryListenerLogger]
        default=True #RMQLogger()

#Config for the FileProcessor to read in pending files for work
[FileProcessor]
        my_id=ffe-file-processor-listener

        host=[redacted]
        user=[redacted]
        passwd=[redacted]

        #RabbitMQConnection Additional Config
        exchange=ffe.pending
        auto_connect=True
        rmq_logger=FileProcessorLogger
        verbosity=11
        queue=pending_files

[FileProcessorLogger]
        default=True

#Config for the FileProcessorOutbount RabbitMQ Writing of records
[FileProcessorOutbound]
        my_id=ffe-file-processor-outbound
        #RabbitMQ Config
        host=[redacted]
        user=[redacted]
        passwd=[redacted]

        #RabbitMQConnection Additional Config
        exchange=records.raw
        auto_connect=True
        rmq_logger=FileProcessorOutboundLogger
        verbosity=11
        #Queue is auto determined based on subdirectory
        queue=

[FileProcessorOutboundLogger]
```

```
        default=True

[SampleAddHeaders]
        # Bool saying this section defines a special rule
        # File Processor Special Rule for File Processor
        FP_SPECIAL_RULE=True
        # Which consumer it applies to
        RULE_TARGET=NojCSVParser
        # Type of rule, AddHeaders, InjectDateTime
        RULE_TYPE=AddHeaders
        # So far just ValueInPath which is a %MATCH_VALUE% match on
        filepath
        RULE_MATCH_TYPE=ValueInPath
        MATCH_VALUE=sample-missingheaderfiles-
        # Black list items
        RULE_EXMATCH_TYPE=ValueInPath
        # Exclusion example
        EXMATCH_VALUE=missingheaderssubdir-raw
        # Comma separated list of headers to apply
        HEADERS=id,name,compass_raw_x,compass_raw_y,compass_raw_z #etc

[SPCInjectDT]
        FP_SPECIAL_RULE=True
        # Inject DateTime will inject a datetime field of DT_FIELD into the
        payloads
        RULE_TYPE=InjectDateTime
        RULE_TARGET=NojCSVParser
        RULE_MATCH_TYPE=ValueInPath
        MATCH_VALUE=sample-
        DT_FIELD=_ffp_dt

[SkipHeaderLine]
        FP_SPECIAL_RULE=True
        # Skip the first row- this rule is commonly used with AddHeaders to
        rewrite the file headers
        RULE_TYPE=SkipFirstRow
        RULE_TARGET=NojCSVParser
        RULE_MATCH_TYPE=ValueInPath
        MATCH_VALUE=sample-missingheaderfiles-
```

CNC COMMAND LINE UTILITY

Our CnC package leverages the capability of open source to simplify deployment of SAS ESP and models built in SAS ESP Studio, smoothing the process of taking models from development to production. Much of its commands are wrapped dfesp_xml_client commands in combination with standard Linux commands for parsing and readability for network health.

LOGIC ENGINE

SAS ESP can process stateful data, but it can be problematic when state needs to be dynamically defined. When an element such as "*the time to finish a work order*" varies, SAS ESP cannot use it's time-based mechanism to mark and track the state of the data. We can however build a logic engine with open source language that uses a set of rules to dynamically define the state of incoming data. It then translates this information to a format that can be understood and digested by SAS ESP. This enables you to use the stateful feature of SAS ESP with any data that needs to be dynamically assigned or labeled.

RABBITMQ HARDWARE AND SOFTWARE

RabbitMQ is an open source message-broker software project (sometimes called message-oriented middleware) that originally implemented Advanced Message Queuing Protocol (AMQP) and has since been extended with a plug-in architecture to support Streaming Text Oriented Messaging Protocol (STOMP), Message Queuing Telemetry Transport (MQTT), and other protocols. In our example, we used RabbitMQ Version 3.7.9. The hardware requirements are relatively small and we utilized a (1) CPU server with 4GB of memory.

The following source code is an example for the sensor records that would come from the ShakeBoxes and be pushed into RabbitMQ:

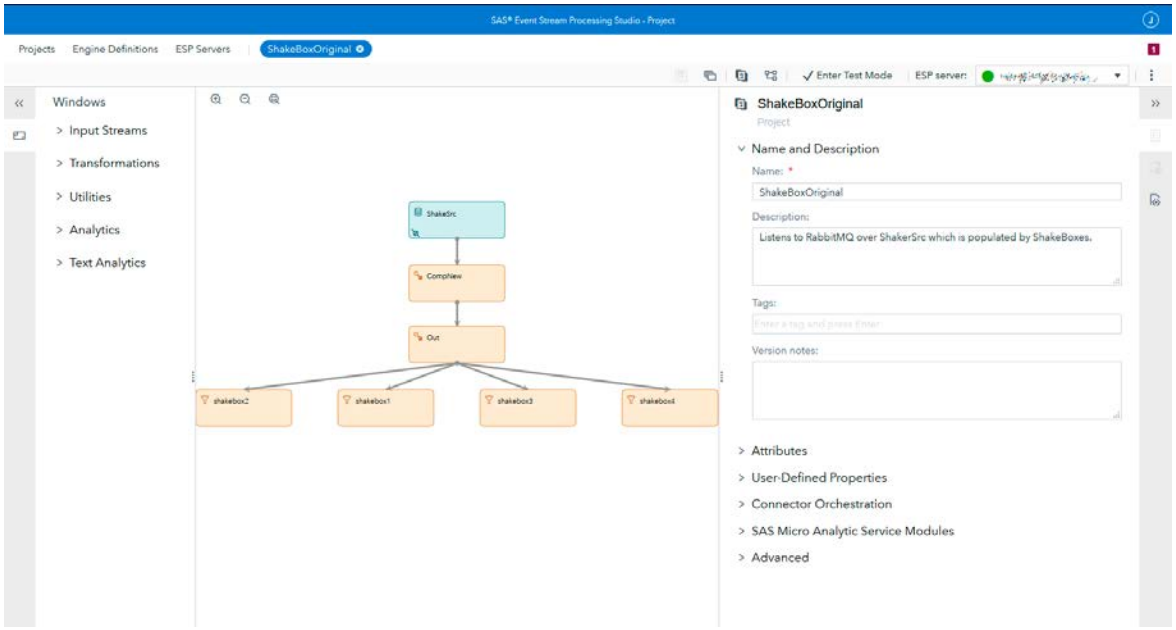
```
import sys, os, datetime
import configparser, json
import time

from sense_hat import SenseHat
import pika

from util.RabbitMQConnection import RabbitMQConnection, RMQLogger

class SensorRecord(object):
        def __init__(self, config_section, display=None):
                """Takes in:
                        config_section - env file [section] to load RMQ config
                        hwid - id to attach to rmq events to identify device
                """
                #SenseHat
                self.sense = SenseHat()
                self.display = display
                #RMQ connection
                self.rmq = False

                #Kill switch- set to False while running will stop the run
loop
                self.running = False

                config = configparser.ConfigParser()
                config.read(os.path.join('..','.env'))
                self.counter = 0
                self.count = 0
                #RMQ connection Details
                self.con_details = {
                        'user': config[config_section]['user'],
                        'passwd': config[config_section]['passwd'],
                        'host': config[config_section]['host'],
                        'exchange': config[config_section]['exchange'],
                        'queue': config[config_section]['queue'],
                        'auto_format': False,
                        'my_id': config[config_section]['hwid'],
                        'auto_connect': True,
                        'rmq_logger': RMQLogger(),
                        'verbosity': 111,
                }
                #ID of the machine
                self.hwid = config[config_section]['hwid']

        def connectRMQ(self):
                """
```

```
                        connct to the RMQ server
                """
                try:
                        self.rmq = RabbitMQConnection(**self.con_details)
                except:
                        self.count += 1
                        if self.count > 5:
                                raise
                        time.sleep(1)
                        self.rmq_error("Error connecting")

        def getReadings(self):
                """
                        Return all the SenseHat sensor readings
                """
                record = {}
                temp = self.sense.get_temperature()
                record['temperature'] = temp
                humid = self.sense.get_humidity()
                record['humid'] = humid
                pressure = self.sense.get_pressure()
                record['pressure'] = pressure
                self.sense.set_imu_config(True, True, True)
                raw = self.sense.get_compass_raw()
                record['compass_raw_x'] = raw['x']
                record['compass_raw_y'] = raw['y']
                record['compass_raw_z'] = raw['z']
                self.sense.set_imu_config(True, True, True)
                accel = self.sense.get_accelerometer_raw()
                record['accel_x'] = accel['x']
                record['accel_y'] = accel['y']
                record['accel_z'] = accel['z']
                self.sense.set_imu_config(True, True, True)
                orientation = self.sense.get_orientation_degrees()
                record['pitch'] = orientation['pitch']
                record['roll'] = orientation['roll']
                record['yaw'] = orientation['yaw']
                self.sense.set_imu_config(True, True, True)
                north = self.sense.get_compass()
                record['north'] = north
                pressure = self.sense.get_pressure()
                self.counter+=1
                if self.counter%100==0:
                        print("SenseHat Active (Pressure: "+str(pressure)+") -
" + str(self.counter) + " Records Sent")
                record['pressure'] = pressure
                return record

        def kill(self):
                """
                        Stop recording
                """
                self.running = False
        def rmq_error(self, message='Lost Internet Connection'):
                print("Lost RMQ Connection")
                if self.display:
                        self.display.commandScreen(message)
                        self.display.please_wait_text = "trying to reconnect"
```

```
                        self.display.setMode("PleaseWait")
                        self.running=False
                        time.sleep(10)
                        self.rmq = None
                        self.start()
        def start(self):
                """
                        Start the sensor recording
                """
                if not self.rmq:
                        self.connectRMQ()
                        self.count=0
                self.display.setMode("idle")
                self.running = True
                while self.running:
                        record = self.getReadings()
                        record['hwid'] = self.hwid
                        record['dt'] = str(datetime.datetime.now())
                        try:
                                self.rmq.broadcast(json.dumps(record),
routing_key='streamtest')
                        except pika.exceptions.AMQPHeartbeatTimeout:
                                self.rmq_error()
                                return
                        except pika.exceptions.StreamLostError:
                                self.rmq_error()
                                return
                        except:
                                self.count +=1
                                if self.count > 10:
                                        raise
                                self.rmq_error('unknown error will reconnect in
10sec')

if __name__ == '__main__':
        recorder = SensorRecord("RMQSensorConfig", "shakebox1")
         recorder.start()
```

SAS ESP HARDWARE AND SOFTWARE

For our example configuration, we utilized SAS Viya version 3.4 and SAS ESP version 5.2. We utilized a multiple server configuration for the SAS ESP and SAS Viya software. The SAS Viya server consisted of an (8) CPU server with 128GB of memory with (2) NVMe SSD drives of 300GB each while the SAS ESP server was configured to be (8) CPU server and 128GB of memory without the local SSD storage. Note that the SAS ESP requirements are significantly smaller and in fact, the SAS ESP can be installed on the Raspberry PI devices themselves for IoT Edge computing!



Display 2. SAS ESP 6.2 Studio Screenshot for ShakeBox Project

The following source code is an example of a SAS ESP project listening to RabbitMQ and pulling in the sensor data from the ShakeBoxes:

```
<?xml version="1.0"?>
<project heartbeat-interval="1" pubsub="auto" threads="1"
name="ShakeBoxOriginal">
   <description>
      <![CDATA[
         Listens to RabbitMQ over ShakerSrc which is populated by ShakeBoxes.
      ]]>
   </description>
   <metadata>
      <meta id="studioUploadedBy">sas_username</meta>
      <meta id="studioUploaded">1581432127439</meta>
      <meta id="studioModifiedBy">sas_username</meta>
      <meta id="studioModified">1581956477059</meta>
      <meta id="layout">
{"cq1":{"CompNew":{"x":470,"y":175},"Out":{"x":470,"y":295},"ShakeSrc":{"x":4
70,"y":50},"shakebox1":{"x":330,"y":420},"shakebox2":{"x":50,"y":420},"shakeb
ox3":{"x":610,"y":420},"shakebox4":{"x":890,"y":420}}}</meta>
   </metadata>
   <contqueries>
     <contquery name="cq1">
        <windows>
           <window-source pubsub="true" name="ShakeSrc" insert-only="true"
index="pi_EMPTY">
              <schema>
                 <fields>
                    <field name="temperature" type="double"/>
                    <field name="compass_raw_x" type="double"/>
                    <field name="accel_x" type="double"/>
                    <field name="yaw" type="double"/>
                    <field name="accel_y" type="double"/>
                    <field name="roll" type="double"/>
                    <field name="humid" type="double"/>
                    <field name="dt" type="stamp" key="true"/>
```

```
        <field name="pressure" type="double"/>
        <field name="compass_raw_z" type="double"/>
        <field name="hwid" type="string" key="true"/>
        <field name="north" type="double"/>
        <field name="pitch" type="double"/>
        <field name="accel_z" type="double"/>
        <field name="compass_raw_y" type="double"/>
    </fields>
</schema>
<connectors>
    <connector name="RMQ" class="rmq">
        <properties>
            <property name="type">
                <![CDATA[
                    pub
                ]]>
            </property>
            <property name="dateformat">
                <![CDATA[
                    %Y-%m-%d %H:%M:%S
                ]]>
            </property>
            <property name="buspersistence">
                <![CDATA[
                    true
                ]]>
            </property>
            <property name="buspersistencequeue">
                <![CDATA[
                    true
                ]]>
            </property>
            <property name="rmquserid">
                <![CDATA[
                    sampleusername
                ]]>
            </property>
            <property name="rmqpassword">
                <![CDATA[
                    samplepassword
                ]]>
            </property>
            <property name="rmqhost">
                <![CDATA[
                    samplehost
                ]]>
            </property>
            <property name="rmqport">
                <![CDATA[
                    5672
                ]]>
            </property>
            <property name="rmqexchange">
                <![CDATA[
                    sampleexchange
                ]]>
            </property>
            <property name="rmqtopic">
```

```
                <![CDATA[
                    sampletopic
                ]]>
            </property>
            <property name="rmqtype">
                <![CDATA[
                    json
                ]]>
            </property>
            <property name="urlhostport">
                <![CDATA[
                    37752
                ]]>
            </property>
        </properties>
    </connector>
</connectors>
</window-source>
<window-compute pubsub="true" name="CompNew" index="pi_EMPTY">
<schema>
    <fields>
        <field name="id" type="int64"/>
        <field name="temperature" type="double"/>
        <field name="compass_raw_x" type="double"/>
        <field name="accel_x" type="double"/>
        <field name="yaw" type="double"/>
        <field name="accel_y" type="double"/>
        <field name="roll" type="double"/>
        <field name="humid" type="double"/>
        <field name="dt" type="stamp" key="true"/>
        <field name="pressure" type="double"/>
        <field name="compass_raw_z" type="double"/>
        <field name="hwid" type="string" key="true"/>
        <field name="north" type="double"/>
        <field name="pitch" type="double"/>
        <field name="accel_z" type="double"/>
        <field name="compass_raw_y" type="double"/>
        <field name="player_id" type="int64"/>
        <field name="hwstatus" type="string"/>
    </fields>
</schema>
<output>
    <field-expr>
        <![CDATA[
            0
        ]]>
    </field-expr>
    <field-expr>
        <![CDATA[
            temperature
        ]]>
    </field-expr>
    <field-expr>
        <![CDATA[
            compass_raw_x
        ]]>
    </field-expr>
    <field-expr>
```

```
                    <![CDATA[
                        [… NEXT SENSOR PARAMTER, etc. truncated for brevity]
                    ]]>
                </field-expr>

:

                <field-expr>
                    <![CDATA[
                        0
                    ]]>
                </field-expr>
                <field-expr>
                    <![CDATA[
                        "TEST"
                    ]]>
                </field-expr>
            </output>
        </window-compute>
        <window-compute pubsub="true" name="Out" index="pi_EMPTY">
            <schema>
                <fields>
                    <field name="id" type="int64"/>
                    <field name="temperature" type="double"/>
                    <field name="compass_raw_x" type="double"/>
                    <field name="accel_x" type="double"/>
                    <field name="yaw" type="double"/>
                    <field name="accel_y" type="double"/>
                    <field name="roll" type="double"/>
                    <field name="humid" type="double"/>
                    <field name="dt" type="stamp" key="true"/>
                    <field name="pressure" type="double"/>
                    <field name="compass_raw_z" type="double"/>
                    <field name="hwid" type="string"/>
                    <field name="north" type="double"/>
                    <field name="pitch" type="double"/>
                    <field name="accel_z" type="double"/>
                    <field name="compass_raw_y" type="double"/>
                    <field name="player_id" type="int64"/>
                    <field name="hwstatus" type="string"/>
                    <field name="boxnum" type="int64"/>
                </fields>
            </schema>
            <output>
                <field-expr>
                    <![CDATA[
                        id
                    ]]>
                </field-expr>
                <field-expr>
                    <![CDATA[
                        temperature
                    ]]>
                </field-expr>
                <field-expr>
                    <![CDATA[
                        compass_raw_x
                    ]]>
```

```
                    </field-expr>
                    <field-expr>
                        <![CDATA[
                            [… NEXT SENSOR PARAMTER, etc. truncated for brevity]
                        ]]>
                    </field-expr>

:

                    <field-expr>
                        <![CDATA[
                            tointeger(replace(hwid,"shakebox",""))
                        ]]>
                    </field-expr>
                </output>
            </window-compute>
            <window-filter pubsub="true" name="shakebox2" index="pi_EMPTY">
                <expression>
                    <![CDATA[
                        boxnum==2
                    ]]>
                </expression>
            </window-filter>
            <window-filter pubsub="true" name="shakebox1" index="pi_EMPTY">
                <expression>
                    <![CDATA[
                        boxnum==1
                    ]]>
                </expression>
            </window-filter>
            <window-filter pubsub="true" name="shakebox3" index="pi_EMPTY">
                <expression>
                    <![CDATA[
                        boxnum==3
                    ]]>
                </expression>
            </window-filter>
            <window-filter pubsub="true" name="shakebox4" index="pi_EMPTY">
                <expression>
                    <![CDATA[
                        boxnum==4
                    ]]>
                </expression>
            </window-filter>
        </windows>
        <edges>
            <edge target="CompNew" source="ShakeSrc"/>
            <edge target="shakebox2" source="Out"/>
            <edge target="shakebox1" source="Out"/>
            <edge target="shakebox3" source="Out"/>
            <edge target="shakebox4" source="Out"/>
            <edge target="Out" source="CompNew"/>
        </edges>
    </contquery>
</contqueries>
</project>
```

SERVERLESS WEB INTERFACE

Hardware

The Web Interface is a serverless, single-page application of modular Javascript widgets built using ReactJS. It can run anywhere that is CORS compliant with the websocket connection to the SAS ESP server. Because the code is compiled, minified, and optimized prior to deployment, it has significantly less overhead then the SAS ESP Streamviewer interface. Based on our estimations, the difference is approximately one-seventh of the memory overhead and a fraction of the file dependencies when loaded by the end user.

Software

Serverless Single Page Application (Web Technologies: JavaScript, HTML, CSS) using open source:

- ReactJS, create-react-app for bootstrap
- Sass (for css)
- Webpack for compilation
- Npm for JavaScript package management
- Recharts for charting library
- AntDesign for initial Layout and style but the ShakeBox game is all custom
- Native JavaScript WebSocket

Display 3. Serverless Web Interface Screenshot

HOW IT ENHANCES SAS SOFTWARE

This solution unites the SAS Platform with open source to enhance SAS ESP in five main ways: data consumption/processing, data visualization, software stability, data quality management, server stability.

The solution enhances SAS ESP's data consumption capabilities with a flat file processor. In a real life use case, a customer needed their machine output data in the form of CSV files. On its own, SAS ESP could not process the CSV files as events, which meant the customer

couldn't perform real-time analytics on their IoT data. The customer was unable to commit the time and resources involved in developing and maintaining an additional middleware layer on top of their existing system because it would have required an update every time there was a change with their ERP system, CRM, and other technologies. The flat file processor that was designed seamlessly bridged the gap between their existing systems and the SAS ESP solution. It gave the customer the flexibility to add and modify their system in the future without fear of breaking their ability to read CSV files into SAS ESP as events. The flat file processor also provides failover capabilities and efficient buffering, the latter being why it has such good performance.

While SAS ESP already outpaces any other comparable product in its sheer number of data source adapters, our added utility focuses in on a common use case and provides developers with a flexible environment. The logic engine makes SAS ESP more useful for developers by adding the ability to dynamically control state and retention.

Our sample bundle also enhances the visualization capabilities of SAS ESP. It allows you to create report via tools that are available in SAS ESP, SAS Studio, and SAS Viya. By delivering customizable, clean visualizations, this solution empowers developers to turn their real-time IoT streaming data into intelligence, a core component of SAS' mission and value.

A key requirement for many organizations is often stability and failover. This can occur from server outages, or when the development team makes changes without consideration of related impact. In the IoT realm, data quality and consistency can be an issue. For example, if incoming data is missing, has gaps, breaks in connectivity, or there are data errors that are sent over from machines or ERP/CRM or other 3rd party systems, it can break the whole process. Since we are limited and we cannot currently utilize an "on-the-fly" conversion in SAS ESP, it would shut down the project and fail. To accommodate the variation in the incoming data, a solution is to intercept the data, format it in a way SAS ESP would accept, and fill in any missing fields with placeholder data. Then the data would be passed to SAS ESP for successful processing. By adding this conversion functionality, we are able to address a variety of challenges for bad incoming data.

This solution illustrates the immense value that can be gained when developers combine the SAS Platform with open source. It strengthens the position that SAS with open source is the future of analytics by demonstrating how they can benefit from investing in this winning combination.

BENEFITS TO MARKETS, INDUSTRIES, AND CUSTOMERS

One of the benefits of this solution is that it delivers real-time analytics using SAS ESP in a consumable, visually compelling way. It makes it easy to examine key metrics and turn IoT data into intelligence that can inform action to solve business challenges. While SAS ESP Studio is available for prototyping and building proof of concepts, we found that it may be somewhat limited to various use cases in displaying visualizations. This approach bridges this gap, bringing the full value of SAS ESP to markets, industries, and customers with IoT Analytics initiatives.

One of the other benefits of this solution is that the CnC command line utility simplifies SAS ESP deployment, by smoothing the process of deployment from development to production. Newer versions of SAS ESP have started to incorporate these functions that were not available in earlier versions.

The approach also provides stability and failover. If you have variation in data quality and completeness, there are techniques that allow capture and data intercept that can format it correctly.

Developers who use CSV files to stream IoT data can now leverage SAS ESP to interpret those files as events. They don't have to switch to a different file format or build time- and labor-intensive, complicated middleware applications that need periodic updates on top of their existing systems. The flat file processor acts as a seamless bridge between CSV data and RabbitMQ, converting CSV files into single records that can be read by SAS ESP and analyzed as events with real-time analytics. With our flat file processor, SAS ESP becomes more adaptable to the type of data the customer is collecting.

With the addition of the logic engine, developers can dynamically define state if needed. This allows customers to fully leverage the power of SAS ESP's capabilities while reducing memory overhead through dynamic purging rules.

Lastly, the Serverless Web Interface dashboard is comprised of widgets that can be customized and switched out. This interface can be seamlessly integrated with Streamviewer or operate completely on its own. This simplifies the process of replicating and then customizing the dashboard for numerous projects. Meaning, this solution has a scalable capacity that allows it to be quickly rolled out to new deployments without rebuilding it each time. The Serverless Web Interface is lightweight and flexible. It does not have strict infrastructure requirements, so it can be used in any market, industry, or customer scenario.

CONCLUSION

Through an IoT Analytics use case using simple IoT devices, this solution illustrates how the SAS Platform and open source resources can work together to achieve the best outcome for the end user. When combining the SAS Platform and open source, developers can benefit from the best of both worlds with a custom solution that can solve their unique business problems.

REFERENCES

MDN Web Docs. 2020. "Websocket." Accessed January 15, 2020. https://developer.mozilla.org/en-US/docs/Web/API/WebSocket

Wikipedia. 2019. "Advanced Message Queuing Protocol." Accessed January 15, 2020. https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol

Wikipedia. 2019. "Message-oriented Middleware." Accessed January 15, 2020. https://en.wikipedia.org/wiki/Message-oriented_middleware

Wikipedia. 2019. "MQTT." Accessed January 15, 2020. https://en.wikipedia.org/wiki/MQTT

Wikipedia. 2019. "Streaming Text Oriented Messaging Protocol." Accessed January 15, 2020. https://en.wikipedia.org/wiki/Streaming_Text_Oriented_Messaging_Protocol

Wikipedia. 2020. "Message Broker." Accessed January 15, 2020. https://en.wikipedia.org/wiki/Message_broker

Wikipedia. 2020. "Plug-in (computing)." Accessed January 15, 2020. https://en.wikipedia.org/wiki/Plug-in_(computing)

RECOMMENDED READING

- *SAS Event Stream Processing Documentation*
- *Getting started with RabbitMQ*
- *Getting started with Raspberry Pi - Introduction*

ACKNOWLEDGMENTS

A big thank you to the Pinnacle Solutions team for working so hard on this solution and putting together this customer use case. A special thank you to Scott Koval, Scott Bowman, Dave Foster, and Ryan Johnston for assisting in the development and execution of this project.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Jon Klopfer
Pinnacle Solutions, Inc.
872-225-0505
jon.klopfer@thepinnaclesolutions.com
thepinnaclesolutions.com

Elizabeth McGlone
Pinnacle Solutions, Inc.
1-317-423-9143 ext. 822
elizabeth.mcglone@thepinnaclesolutions.com
thepinnaclesolutions.com

Donald (DJ) L. Penix, Jr.
Pinnacle Solutions, Inc.
1-317-423-9143 ext. 801
dj.penix@thepinnaclesolutions.com
thepinnaclesolutions.com