Embiot – A Small Footprint Embedded Edge Analytics Agent

Alan Clark

alan.d.clark@telchemy.com

IIT RTC Conference October 2020

Outline

- What problem(s) are we trying to solve?
- Embedded Stream Processing
- Architecture
- Implementation
- Practical examples

Edge Analytics

- Why integrate analytics into edge devices?
  - Minimize dependency on network and Cloud resources
  - Reduce volume of data sent to Cloud – less network bandwidth and less Cloud storage
  - Respond faster to detected problems – take local action and generate local alerts
  - Access to larger volume of/ more frequently sampled sensor data, ability to correlate in real time
- Already widely established for autonomous vehicles, smart video surveillance….
- Often implemented as dedicated applications software or analytics libraries for Python, Java, C….

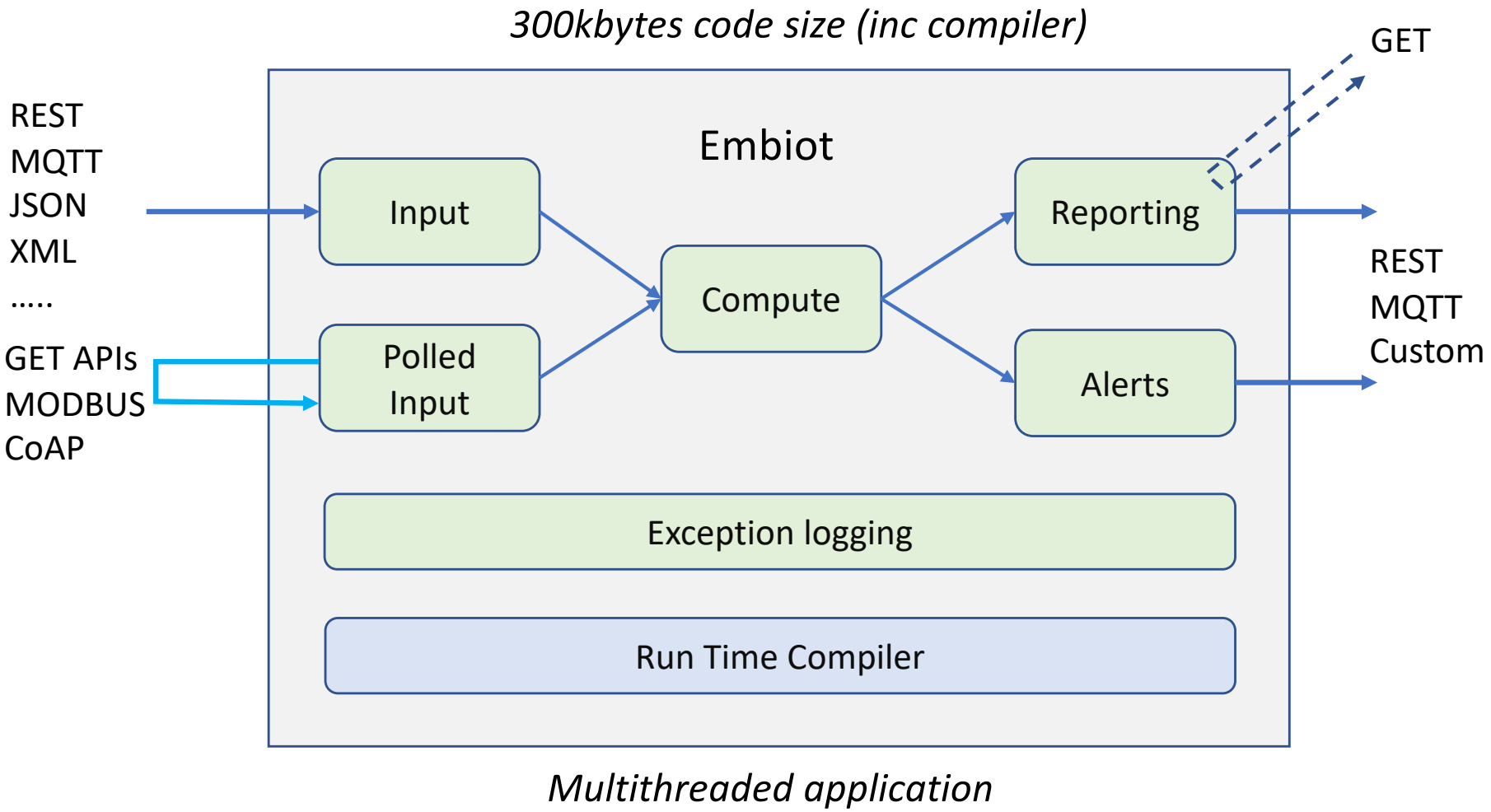
Goals for a general purpose agent

- Make it easy *for the user to* implement real time analytics at the edge/ in the device
  - (Ideally) should not require a CS background
- Handle multiple streams of asynchronous data,
- Should be robust and recover from bad/missing data, reboots, disconnections
- Make it easy to (=minimal configuration) get data in / out from a variety of sources/ formats
- Small footprint (~100-300kbytes) to allow integration into a wider range of devices and smart sensors

Approach

- Stream processing / compute flow
  - Inherently handles many asynchronous data streams
- Comprehensive function library
- Integrated runtime compiler
  - Makes prototyping and experimentation quick and easy
- Range of input methods
  - Listen, Polled, Subscribe-Publish….
  - Ability to “figure out” what the input data format is
- Range of output methods
  - Minimize configuration needed to generate periodic or alert reports

Implementation

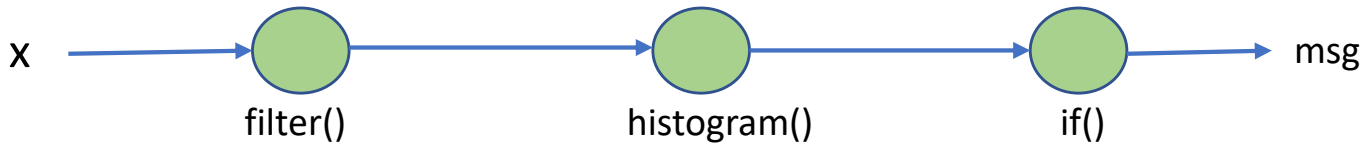


Embiot language

- Declarative programming language
  - Maps well into a stream processing/ flow computing model
- High level constructs for
  - Servers, brokers, gateways
  - Devices and Device Classes
  - Reports
- Metrics – typed, supports heirarchic naming
  - All metrics objects have “.” properties, last value, timestamp, errors, timouts, interval……
- Rules – typical algebraic expressions with math/ stats/ logic/ string functions
  - Automatic type conversion (e.g. string to float)

Compute - model

Rule: h = histogram(filter(x, -0.1, -0.1, 0.4, -0.1, -0.1), 50, 0, 10)
Rule: msg = if(6 < h.median < 10, “too high”, “normal”)

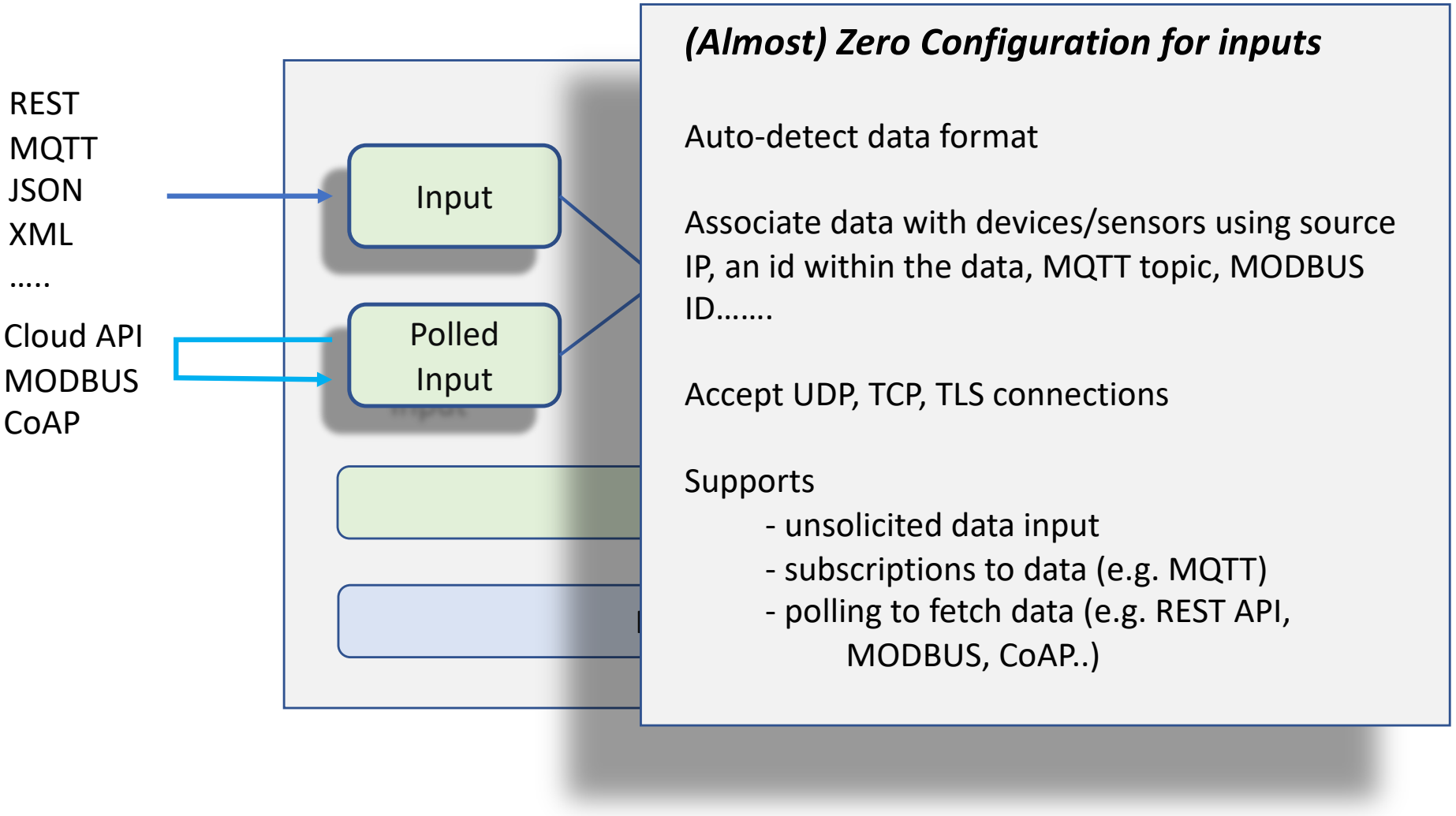


All nodes have exception handling and logging, e.g. divide by zero, not updated within time interval…

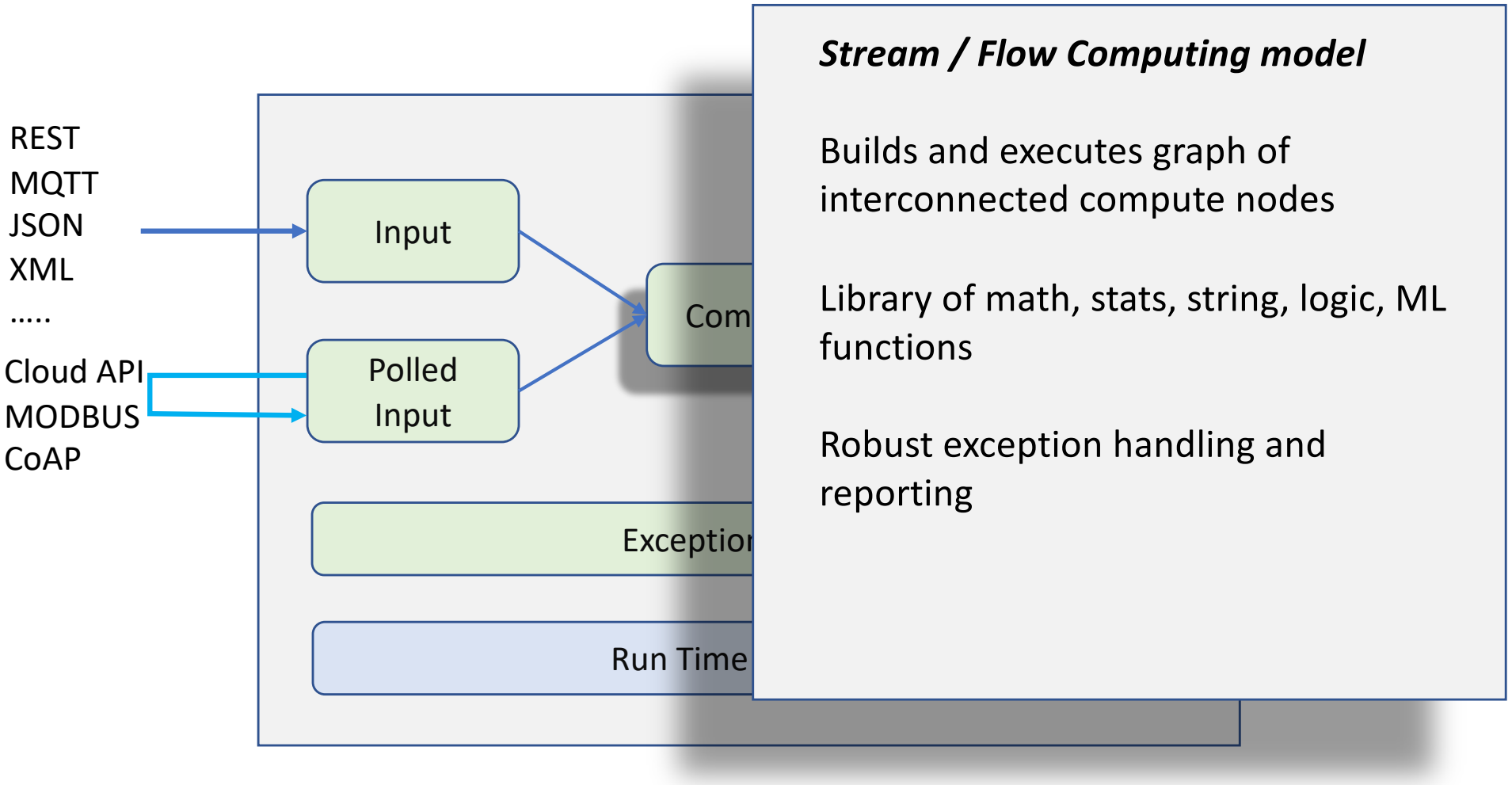
Function library

- Typical math/ stats/ string functions
- Standard Deviation, Histogram (auto scaling)
- Range normalization, scaling and sampling
- FFT, Autocorrelation, Filter
- Neurons, convolutional neurons
- Probabilistic and fuzzy logic
- Application specific functions such as volume()
- Logic and bit manipulation
- Time format conversion
- Geocoordinate manipulation

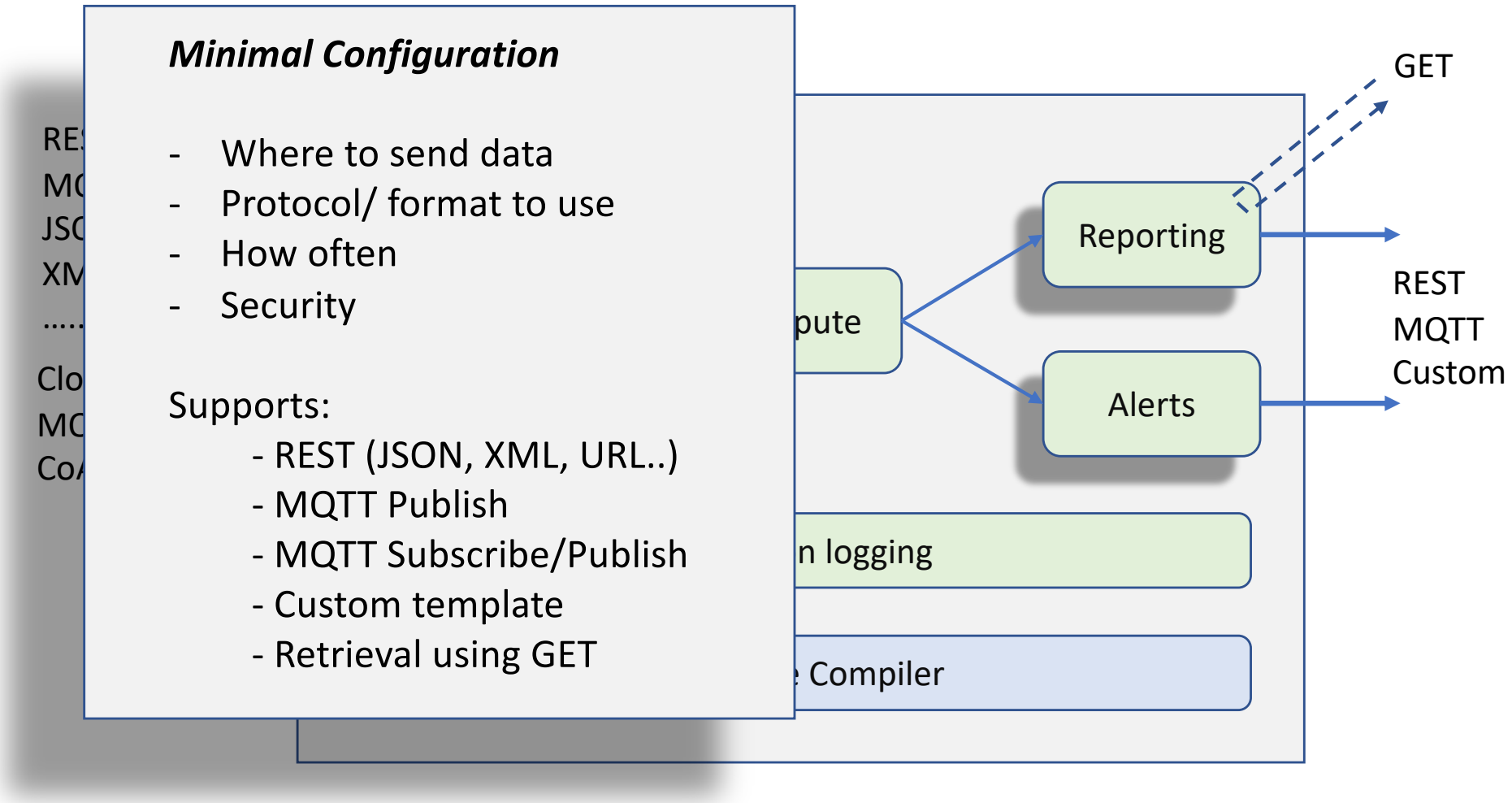
Input


REST
MQTT
JSON
XML
.....
Cloud API
MODBUS
CoAP
Input
Polled Input
(Almost) Zero Configuration for inputs
Auto-detect data format
Associate data with devices/sensors using source IP, an id within the data, MQTT topic, MODBUS ID…….
Accept UDP, TCP, TLS connections
Supports
- unsolicited data input
- subscriptions to data (e.g. MQTT)
- polling to fetch data (e.g. REST API, MODBUS, CoAP..)

Compute


REST
MQTT
JSON
XML
.....
Cloud API
MODBUS
CoAP
Input
Polled Input
Com
Exceptio
Run Time
Stream / Flow Computing model
Builds and executes graph of interconnected compute nodes
Library of math, stats, string, logic, ML functions
Robust exception handling and reporting

Output


Minimal Configuration
- Where to send data
- Protocol/ format to use
- How often
- Security
Supports:
- REST (JSON, XML, URL..)
- MQTT Publish
- MQTT Subscribe/Publish
- Custom template
- Retrieval using GET
GET
Reporting
Alerts
REST
MQTT
Custom
logging
Compiler

Example 1

```
input_tcp_port: 8898

report_name: myreport
report_enable: true
report_host: somehost.com
report_port: 80
report_type: PUT
report_encoding: JSON
report_interval: 4
report_metrics: r, s

metric: x, float, input
metric: f, float
metric: r, float
metric: s, string[128]
rule: f = fft( x, 2048, 5000 )
rule: r = f.harmonics[3] / f.harmonics[1]
rule: s = if( r > 0.3, “high 3rd harmonic”, “low 3rd harmonic” )
```

Receives x on port 8898 in (almost) any format

Calculates fft and 3rd harmonic ratio

Generate text message based on 3rd harmonic ratio

Every 4 seconds generates a report as an HTTP PUT with JSON encoded data

Example 2

```
mqtt_broker: broker
mqtt_broker_hostname: [host]
mqtt_broker_port: 1883
mqtt_broker_username: [username]
mqtt_broker_password: [password]
```

Automatically subscribes to data from two MQTT enabled sensors

Receives MQTT Publish messages resulting from the Subscribes

```
device_class: mqttdev
device_protocol: mqtt
object: timestamp, input, string[32]
object: accelX, input, float
object: accelY, input, float
object: accelV, float
```

Associates data in messages with device/ object

- mqttA/accelX
- mqttA/accelY
- mqttB/accelX
- mqttB/accelY

```
device: mqttA, mqttdev, broker, [Topic string]
device: mqttB, mqttdev, broker, [Topic string]
```

Uses wildcard notation to replicate a rule for each device

```
rule: */accelV = sqrt( sqr( */accelX ) + sqr( */accelY) )
```

Advantages

- Robust real time embedded application – designed to run indefinitely
- Quick and easy to program
- Device/Device Class makes it easy to handle large numbers of identical devices
- Less computationally efficient than compiled C/Java but provides more high level functions (e.g fft, acf, filter, histogram, neuron…) to improve efficiency
- Tested on 500MHz ARM6 through to Intel quad core with up to 70,000 metrics/ rules and over one million metrics per second

Summary

- Very flexible, small footprint edge analytics agent
- Does not require a CS or Data Science background to create a real time analytics application
- Uses stream processing/ compute flow model to robustly handle asynchronous data
- Future work
  - Support external modules to enable dedicated compute intensive functionality to be used from within Embiot
  - Smaller version (50kbytes or less)
  - Expand range of ML functionality