DIANOMIC

Edge Application Development FogLAMP as a Rapid Development Tool

For No-Code Engineers to Python or C/C++ Programmers

FogLAMP is an open scalable flexible data collection and forwarding platform, collecting data from any device and forwarding it to any destination. Going far beyond data acquisition, FogLAMP's data processing, filtering, signal processing, and notification services make it a platform that allows application development using a model of connecting off the shelf components together to perform complex operations. In addition, a REST API allows the data gathered by FogLAMP to be viewed and managed in other applications on the edge, ISA95 systems or clouds.

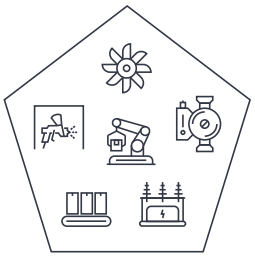
This paper highlights some of the techniques that can be used to build applications on the edge either within the FogLAMP framework itself or by connecting to the API offered by FogLAMP. Building no-code applications, provisioning, updating, deleting, configuring, back up, securing and scaling an industry 4.0 edge using FogLAMP is done by using FogLAMP Manager. Please read An Introduction to FogLAMP Manager for more information.

Basic Components of an Industry 4.0 Edge

The four physical components of a 4.0 edge are assets, data sources, edge devices and integrations. Assets are the objects being monitored. Data Sources monitor those assets and create the data for monitoring those assets. Edge Devices are the compute engines where FogLAMPs are installed and operate. Integrations are the data destinations generally ISA95 systems and/or the cloud.

Industry 4.0 Edge Physical Components

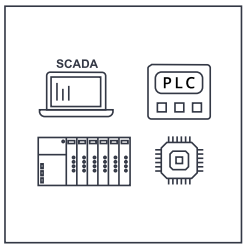
Assets



Object(s) being monitored

- Fans
- Pumps
- Transformers
- Floors
- Paint booths
- Robots
- etc.

Source Devices



Monitors assets and creates data

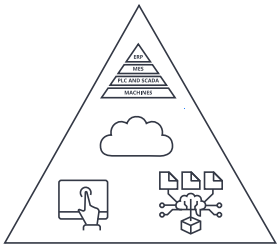
- SCADA, PLC, DCS
- Aggregating head end system
- Physical proxy for sensors
- Multiplexer connecting sensors
- Sensors
- Sometimes: Source Device embedded in Asset

Edge Device

Runs FogLAMP

- Gateway
- VM
- Container
- Sometimes: Edge Device embedded in Asset, Source Device or External System

Integrations



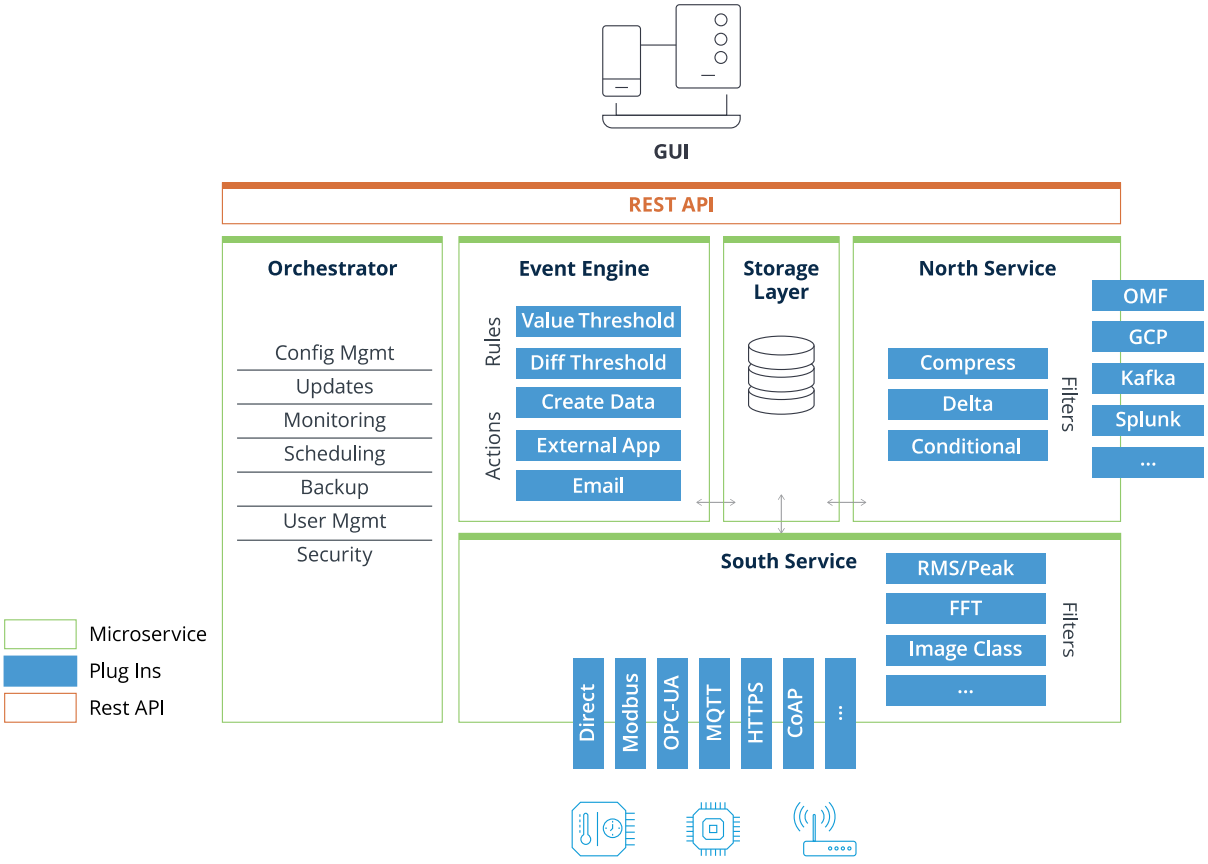
Destination to export data from FogLAMPs:

- ISA95 System
- Cloud
- ML System
- HMI
- Any

DIANOMIC

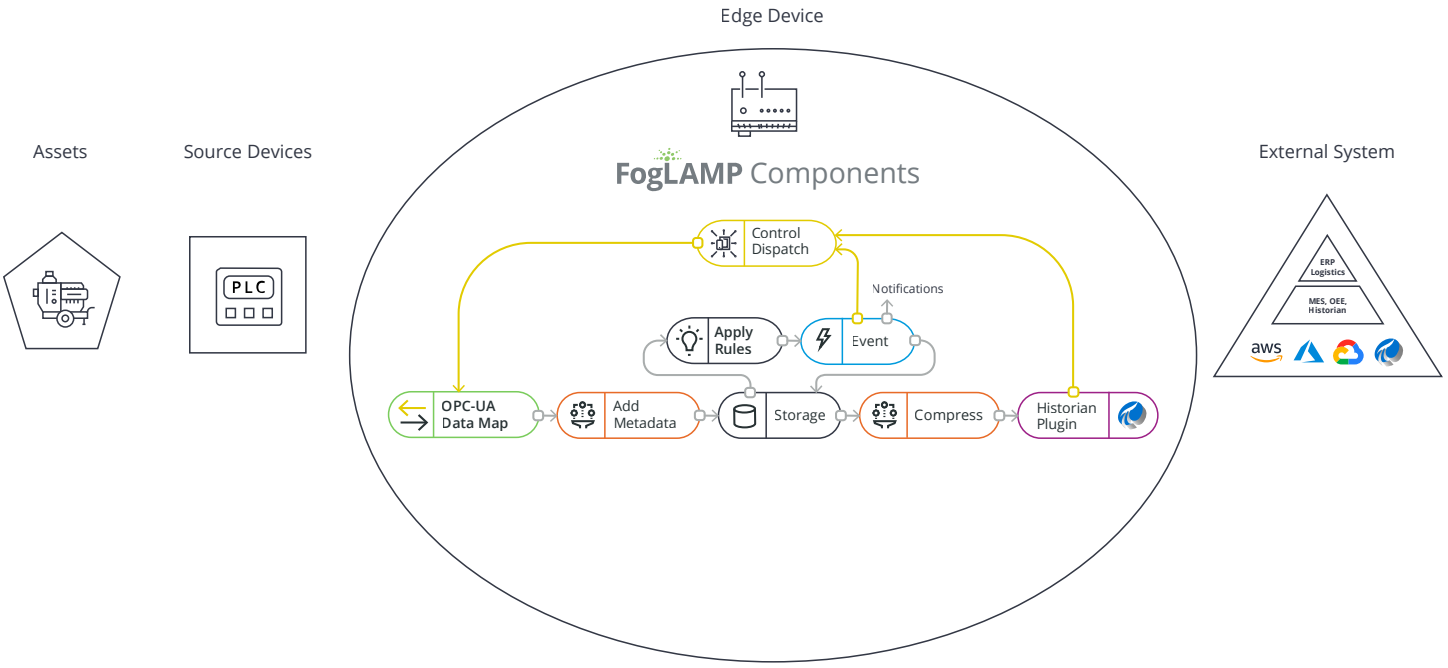
The Processing Model

It is important to understand the processing model of FogLAMP before commencing development of a new application or data pipeline. The basic flow begins with readings entering FogLAMP via the south plugin, getting converted to a common JSON format in the storage layer and exiting via the north plugin.



Readings can also be created in a filter pipeline or as a result of a notification. Knowing where in the lifecycle of a reading traversing through FogLAMP it may be processed and what off the shelf facilities exist for processing the data is recommended. These readings then pass through elements of the FogLAMP system before eventually being sent north to one or more external systems or being removed by a filter somewhere within a filter pipeline.

Basic FogLAMP pipelines include: Data Ingest, Ingress Filters, Storage, Rules, Notifications, Events, Egress Filters And Data Stream Integrations. Bi-directional control can be used for non real-time control.



Common Types of Application Pipelines

Data Collection and Processing

- Unit conversion
- Synchronize data
- Signal processing
- Add metadata
- Remove or reduce data
- ML computer vision

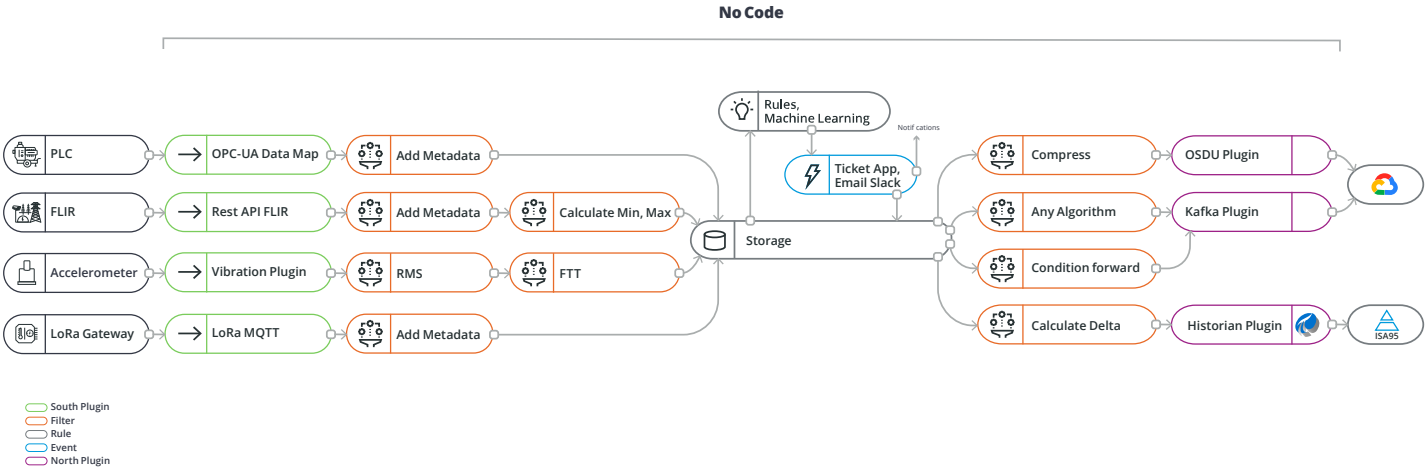
Data Analysis and Rules

- Simple thresholds
- Evaluate w/ any algorithm
- ML based anomaly detection
- Events
  - Send Alerts
  - Interact w/ other systems
  - Create derivative data

DIANOMIC

Simple Pipeline Example

Complex Pipeline

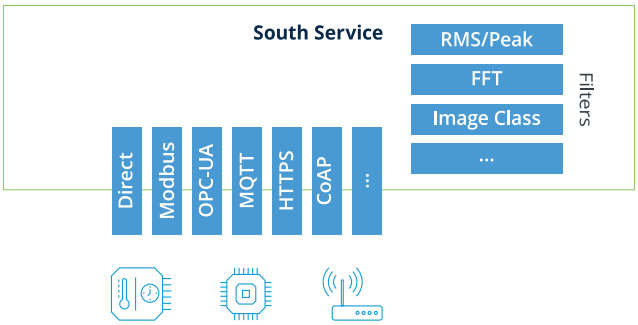


Processing on Ingress

In general data enters FogLAMP via a south service, though there are some exceptions to this that will be made clear as we go further through this paper. These south services use plugins to read from devices or to implement protocols that ingest the data. All the data enters FogLAMP as JSON objects with a timestamp and a set of data points that represent the actual values read. Each south service reads from a single source of data and is independent of every other south service within FogLAMP. The plugins allow the south service to be customized to particular devices or protocols[1] without having to rewrite the framework of the south service for each device. South services always send the data that is read to the FogLAMP storage service. The storage service is a buffering service that is used to collect the data from the various south services before forwarding it to other FogLAMP services.

A south service may also add a set of filters into the processing of the data as it streams from the south service to the FogLAMP storage service. These filters may augment, modify or delete the data that is read from the devices, providing the first opportunity within the FogLAMP data stream to process the data.

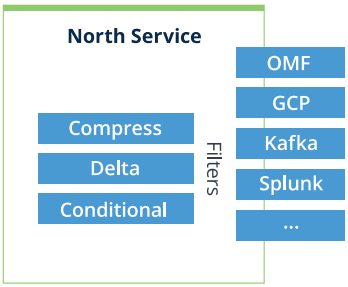
1 For the sake of brevity we will simply refer to devices for the remainder of this discussion, however the points discussed apply equally to protocols and physical devices.



The filters may be arranged in a pipeline, such that the data that is read from the device is fed into the first filter in the pipeline, the data that comes out of that first filter is then fed into the second and so forth until the final filter in the pipeline. The output of the final filter is then sent to the storage service for buffering.

Each filter in the pipeline may output as much or as little data as it desires, there is no requirement that a filter must output a reading for every reading that is input to the filter. This allows the filters to add or remove readings from the stream as well as modifying the data in the stream. The filters themselves are implemented as plugins, written either in C/C++ or Python. A growing number of filters exist, approximately 30 at the time of writing. The collection of filters allows application development to be done without the necessity of coding, however the design of the filter plugin API is such that developing a new filter is not a particularly complex process if it should be required.

Processing on Egress



Data leaves FogLAMP for upstream systems, including other FogLAMP instances using the north task. These tasks have a variety of plugins that are used to match the FogLAMP data to the external systems and to implement the protocols needed to communicate with those systems. Processing may also be done as data is pulled from the storage buffer and before it is processed by the north plugin. This processing works in exactly the same way as the ingress processing, a filter pipeline may be used to process the data. In the case of the north the data that passes through the filters will not be a single asset or a set of assets from a single south service, but rather all the assets that are being sent to the north system.

DIANOMIC

Filter Categories

The filters can be broken down into a number of categories:

- Those that modify single readings by applying a simple operation.
- Filters that add extra data to an individual reading, either static data or data derived from some calculation.
- Filters that add extra readings into the data stream.
- Filters that remove readings from the data stream.
- Filters can replace one or more readings with data computed from the values of those readings.

Simple Modifications Filters

These are perhaps the simplest of all the filter categories. They implement the simplest of functions, such as scaling or adding an offset to the values of the data read from the device. They are useful for converting from one set of units to another to enable direct comparisons of data further upstream. The most obvious of these is the scale filter itself that allows a scale multiplier to be applied to every value within a reading and also a constant offset.

Data Augmentation Filters

Data augmentation filters do not change the existing data, rather they enhance that data by adding new data to the reading. This may be static data or data derived from the values in the reading or the history of the reading.

The simplest augmentation filter that adds static data to the reading is the metadata filter. It adds static items to each reading, typical uses are to tag the data with units, or data regarding the collection point of the unit. An example might be to add a machine location or serial number to the reading to add traceability later in the processing stream.

Another example of adding static data is the OMF Hint filter. This adds hint data to the reading that is later picked up by the outgoing OMF north plugin to guide that plugin as to how to deal with the reading.

Another filter in this class worth highlighting is the expression filter, it allows an arbitrary mathematical expression to be applied to the data values within the reading. The expression may be entered in the configuration of the plugin, allowing for users without any programming experience to modify the values read from a device using complex mathematical formulae. This filter can be used to modify the data, for example making the scales logarithmic or to create new data based on the data in the reading. An example might be to use a voltage and current reading to determine the power consumption of a device.

Data augmentation filters may also work on more than just a single reading, there are many examples of filters that will create new data based on a range of readings using either time or reading count.

Statistics Filter

The statistics filter is designed to accept data from one or more assets and create statistics over specified time intervals, for example to produce the mean, standard deviation and variance for 100 milliseconds samples of the data. The statistics that can be produced are:

- **Mean:** the average of all the values in the time period calculated by adding up all the values and dividing by the number of values.
- **Mode:** the number that appears most often in the time period.
- **Median:** the median is found by sorting all the values in the time period and then choosing the middle number in this sorted set.
- **Minimum:** the minimum value that appears within the time period.
- **Maximum:** the maximum value that appears within the time period.
- **Standard Deviation:** the standard deviation measures the spread of the numbers above and below the mean value.
- **Variance:** the variance is the average of the squared differences from the mean value calculated over the time period.

These statistics values may then be used by downstream filters, stored and sent to the destination system or used within the rules that implement edge notifications.

Fast Fourier Transform Filter

The Fast Fourier Transform filter is designed to examine incoming waveform data and extract frequency based data using the FFT algorithm.

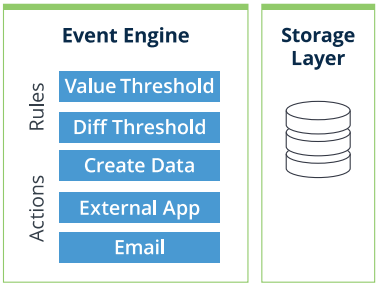
Root Mean Squared Filter

The RMS filter uses a root mean square algorithm to determine the power of a waveform.

Vibration Features Filter

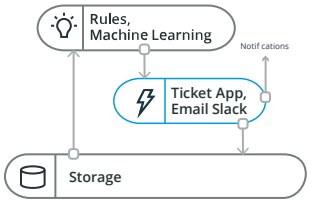
The vibration features filter is designed specifically to handle incoming data that is from a vibration sensor and to extract features that are useful downstream to machine learning or simple rule based notification rules.

The FogLAMP Notification Model



FogLAMP supports an optional microservice that is used to generate notifications or events. The notification service sits logically in the dataflow between the south ingress services and the storage service. It receives the data after it has been through ingress processing in the south services but before it has been processed for egress. As a result any processing done in south services is available to the notification service but not processing done on egress.

The notification processing takes place in parallel to the buffering and forwarding of data via egress services, as a result notification processing happens soon after the data is first read by the FogLAMP device. The only latency involved is the degree of buffering that has been configured into the south service.



The notification service receives data that flows into the FogLAMP buffer and passes this to the various rules that have been configured for evaluation. If a rule is triggered then the delivery plugin associated with the notification is called to perform the action associated with that notification. Both notification rules and notification delivery mechanisms are implemented as plugins to the notification service and are thus extendable in the same way as other services within FogLAMP.

Notification Rules

A notification rule plugin is the evaluator that is used to generate notification, it requests data from the notification server for one or more assets of interest. It may also ask for data to be delivered in a specific way, either as a time window of data with a function applied to it (e.g., an moving average over a 10 second period), or it may request each individual data point to be delivered to it.

The notification service then calls the evaluation entry point of the rule whenever new data is available, the rule may return either a triggered or a cleared status based on the data it received and any processing the rule performed. The rule may also store data within itself for evaluation in future invocations if it desires.

What happens when the rule returns its status depends upon how the notification has been configured, three options are supported:

1. Call the delivery routine if the rule state changes from cleared to triggered between invocations.
2. Call the delivery routine if the rule state is triggered and the delivery has not been called for a specified time.
3. Call the delivery routine if the rule state has transitioned from triggered to cleared.

In all cases it is possible to configure a minimum time between notification deliveries in order to prevent the flooding of the system to which notifications are delivered.

A number of generic rule plugins exist within FogLAMP which allow for the crossing of threshold, rapid changes in data values or expression evaluation. In addition specific rules can be created in Python or C++ to expand upon this functionality.

Notification Delivery

Notification delivery plugins can take action to trigger external systems or feed into FogLAMP itself when a notification is triggered or cleared. The notification is called with the rule state that causes the notification and information associated with the rule and data. Some common delivery mechanisms that are available within FogLAMP include:

- Send a message to a common messaging platform such as Slack, Skype, Google Hangouts, Telegram, etc.
- Send an email message.
- Add an assert into the FogLAMP data stream.
- Execute a Python script.
- Interact with the Amazon Alexa platform to send a voice notification.
- Trigger an action in IFTTT.
- Trigger the delivery to a north side system that is otherwise disabled.
- Alter the internal configuration of FogLAMP, for example increase the collection rate of certain data when a condition becomes true.
- Raise a problem ticket in Jira or Zendesk.

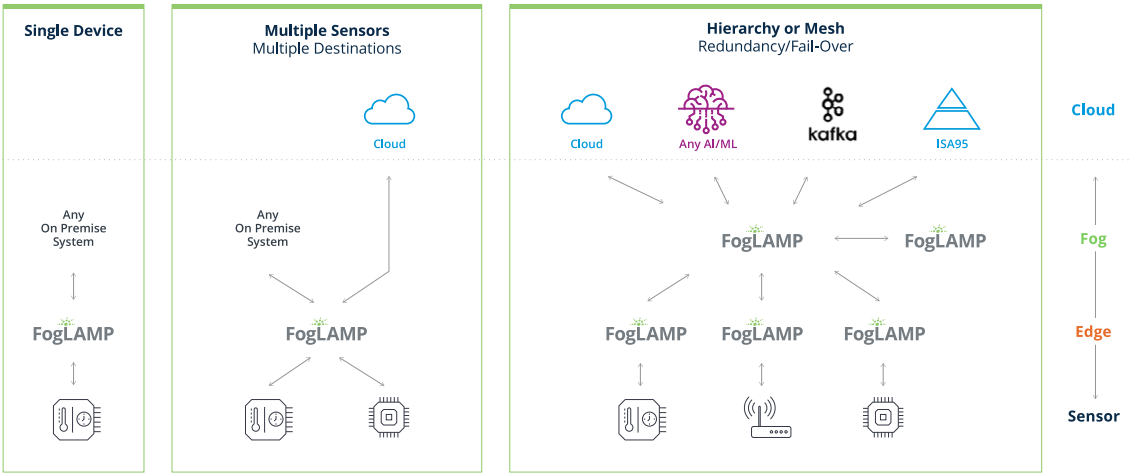
The notification delivery plugins can be easily extended to perform any action desired with relative ease. Adding an asset into the data stream can be useful for audit purposes for triggering specific analytics in one of the destination systems.

Building Applications

Combining the data processing abilities of the filters and the ability of the notifications service to detect events and trigger actions on external systems it is possible to write applications merely by combining these features. Where specific functionality is required it may become necessary to create new plugins to implement that functionality, however this is simpler than creating complete bespoke applications.

Should these facilities not provide sufficient functionality there are three more options available to produce an application that interacts with a FogLAMP system:

- Write a standalone application that uses the FogLAMP data browsing API to extract data from the FogLAMP buffer.
- Write a custom microservice using the FogLAMP libraries that is sent data as it arrives into FogLAMP.
- Write an application that is fed data from the north of FogLAMP using one of the protocols supported by the FogLAMP north plugins.



Standalone Applications

FogLAMP offers a REST API that can be used by any application in order to extract data from the buffer contents of the FogLAMP device. This allows data to be retrieved and processed by an external application for the duration of that data being held in the buffer. Using this mechanism to write applications against FogLAMP devices offers flexibility but has a few disadvantages:

- The application must poll the data as there is no mechanism for an application to be pushed data.
- An application can only access the data up to the point that the data is purged. If an application requires long term access to any of the data it must provide its own mechanism for doing this.

Custom Microservice

A microservice within the FogLAMP environment is relatively easy to produce and has some advantages over writing a standalone application:

- A set of standard libraries is available for creating microservices, interacting with the microservice management API and with the storage system API's.
- Configuration is available and integrated with the FogLAMP configuration management system.
- Microservices can register with the storage service to be delivered updates to all or a subset of the asset readings as they arrive, negating the need to poll for new data.

Microservices can be written in Python or C++ and will be scheduled by FogLAMP once they have been installed into the FogLAMP environment. They also benefit from the FogLAMP monitoring a recovery of microservices should they fail.

North Application

One of the features of FogLAMP is that data can be delivered to multiple north destinations, therefore nothing is lost by adding a second north destination to send data to an application that runs at or near the edge device.

This has the advantage over using the data browsing API that data is sent by FogLAMP without the need to either poll.

Compared to writing a custom microservice there is no longer the need to tightly integrate with FogLAMP. Also, you may use the language of your choice rather than being limited to the languages for which FogLAMP provides libraries.

The choice of communications protocol to use will be dictated by the ease of availability in your chosen language and also the availability of a north plugin in FogLAMP. The HTTP/HTTPS plugin is possibly the simplest candidate if this approach is used. You will need to be able to decode the JSON payloads and be prepared to consume blocks of data that may be large rather than work on small numbers of readings.

Since the connection to FogLAMP is via the north services of FogLAMP, and each north destination has its own north task or service to feed the data, you are free to add custom filter pipelines that pre-process the data for your application without having any impact on the data that is sent to other north destinations.

DIANOMIC

The Edge Application Life-Cycle

Engineering, maintenance, operation, business teams may or may not know how to code. FogLAMP's no-code application environment enables them to write the applications to get the information they need from the edge. However, how do they deploy, operate and manage these applications without creating significant new burdens on ITOps, SecOps and compliance?

FogLAMP Manage is an IIoT life-cycle management system designed to enable the OT workforce with a self-serve scalable industrial 4.0 Edge. OT experts know their own machines, processes and systems. The application life-cycle usually starts with a condition monitoring, predictive maintenance, efficiency, quality, situation awareness or safety opportunity or problem that the OT team uniquely understands how to address. The second phase is the creation of the application. At this point, just like a smartphone, the application needs to get securely provisioned to the correct gateway, VM or container on the edge.