EdgeX Foundry Micro Service System Management

Last updated: 1/9/2018

The California release of EdgeX is expected to contain some new elements in order to support ~~of~~ system management functionality.

This document describes two elements of a proposed system management capability, to be considered for the California release. The first introduces an initial system management service API, which will be incorporated into each existing EdgeX micro service, and required for any new EdgeX micro service. The second describes a new system management agent (referred to as Agent in this document), that uses the API defined in the first element, and could be used to integrate the proposed EdgeX system management service API to alternate management systems~~ and API~~.

Management Service API

The following APIs will be offered by each micro service as part of the system management capability within EdgeX. The service APIs below do not specify additional security parameters or tokens that may be required (depending on security implementations and requirements).

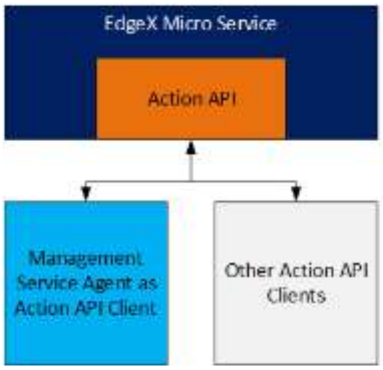
It's assumed that many of the APIs will require some type of authentication/access-control, and depending on sensitivity of information being collected, the Metrics API may require the same. Any such requirements are outside the scope of this proposal, and will be handled by the EdgeX Security Working Group.

Action API

Action API are requests for device information or micro service configuration get/set **operations**, including state changes (operating and administrative states as defined below). Each micro service must have the ability **to** respond to the Action API requests. These requests may come from the Agent or other client**s**.

Commented [Tony Espy1]: You need a noun after API: functions, endpoints, ...

Commented [Tony Espy2]: Not sure what you mean by this? Does this mean every micro service should respond to, but not necessarily implement every endpoint defined by this API?



- Ping the service (no parameters required): a check to see that the micro service is reachable by its known address and is able to respond. Ping is currently implemented by all micro services and used as a health check endpoint by a micro service registry service.

Commented [Tony Espy3]: Same comment about a bullet only on the first item, maybe drop their usage altogether?

Note: today, the micro service registry service is fulfilled by Hashicorp's Consul – an open source registration and configuration software application. As with all of EdgeX, this service may be replaced going forward, but its functionality is considered critical to EdgeX (and is considered part of the Core Services) as well as EdgeX system management capability (and should also be considered part of the System Management Services). Therefore, any replacement of this service must address needs across these architectural areas.

Commented [Tony Espy4]: Suggest bold, and line break after previous <p>

Core Services
Registry & Configuration
System Management Services

Set the micro service's configuration **setting** (aka property) (key or name and new value): set the configuration **setting** for a particular dynamically adjustable setting. For this first iteration of the APIs, the documentation for the service should note which settings are dynamically adjustable and which are not. The dynamic nature of a configuration **setting** may be something that is available via the service API.

Commented [Tony Espy5]: Maybe a better description is "for a setting that supports writes (vs. read-only)."

Note: the terms configuration and property can lead to confusion and sometime have different connotations and meanings within different IT & OT circles. Here, the term configuration refers to a name/value pair that is used by the micro service to replace a software program placeholder (which is a uniquely named key) with a value. The name/value pairs are used by the software at the time the application starts and initializes, but some substitutions of the placeholders (the named keys) may be done each time value is needed in the application – thus any change to the configuration is used immediately by the application.

Get the micro service's admin or operating state. (See https://wiki.edgexfoundry.org/display/FA/Definitions for the formal definition of Admin State).

Set the micro service's admin state (passing either locked or unlocked): set the micro service's administrative state. When set to ***locked***, the micro service should not respond to any other requests from any other service except a request to set the admin state's back to ***unlocked*** (responding with service not available status on any request). (See https://wiki.edgexfoundry.org/display/FA/Definitions for the formal definition of Admin State).

Commented [Tony Espy6]: Including current a query for current admin state?

Also, to be clear, the service will not reply with an error, it'll just drop the request correct?

Note: operating state is not something that should be "set" as it is determined from the actual state of a service or device.

Stop (immediately) the micro service (no parameters required): a request to shutdown the micro service as soon as possible.

"Gracefully" stop the micro service (no parameters required): a request to shutdown the micro service as soon as all existing process requests have been handled.

Get all the micro service's configuration settings (no parameters required): a request for all of the service's configuration key/value pairs (key is the setting name and value is the value of the setting).

Get all the micro service's configurations (no parameters required): request a collection of all the configuration keys.

Get a micro service's configuration for a specific setting (setting key or name as a parameter): get the configuration value for a particular setting.

Get the micro service name (no parameters required): a request for the micro service name as it is known to all of EdgeX specifically to Config/Registry and Core Metadata micro services.

Get the micro service version (no parameters required): presuming that multiple versions of a micro service may exist in the future, a request for the micro service version as it is known to all of EdgeX specifically to Config/Registry and Core Metadata micro services.

There will be a "restart the micro service (no parameters required)" API that is a request to shutdown and then start the micro service as soon as possible. Because a program will not be able to restart itself, implementation of this feature is to be determined. It might be that the Agent (see below) performs a stop of the existing micro service and then a start of a new instance of the micro service. It might be that the request to restart causes the micro service to launch another copy of itself and then shut itself down.

Alert / Notification API

Alerts or notifications are callbacks made by another EdgeX micro service (or other client such as the Agent), for the purposes of notification of a predefined event, such as a change in state or a pre-determined condition. The Agent may request this alert from the micro services so that it can then inform other systems or otherwise react to system problems.

Each micro service must have the ability to retain a list of clients, each client's list of state or condition changes that they want to be notified of, and the callback address to connect to when the change in state or condition happens inside of the micro service. This registry of information must be durable/persistent – that is able to be retrieved and used on failure or restart of the micro service.

Commented [Tony Espy7]: I don't see this as all that useful, especially if services have to register themselves by name both with Consul and metadb.

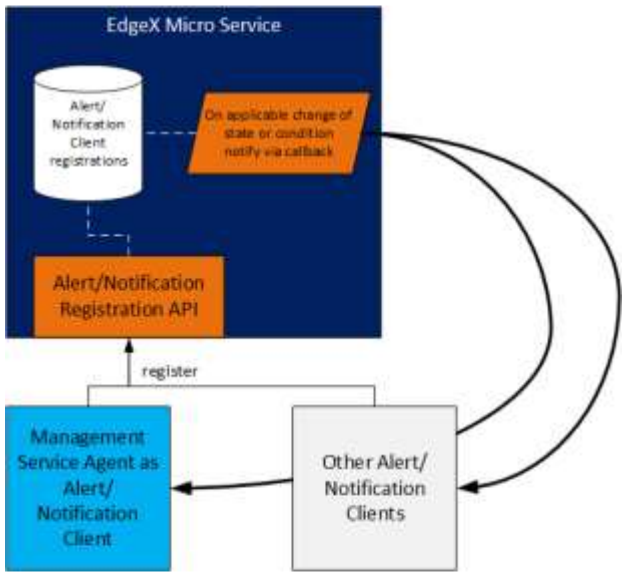
However as previously mentioned, I do like the idea of a service object being returned. Why wouldn't we mimic this on the existing service domain object?

Commented [Tony Espy8]: I'd suggest command, endpoint, function or request. API is the term for the entire collection of functions.

Commented [Tony Espy9]: Will leverage native operating system and/or container functions.

Commented [Tony Espy10]: Not a fan of this approach...

Commented [Tony Espy11]: The other approach is resiliency, where clients notice the service went down and just automatically re-register. It's a simpler solution that removes the requirement for each service to implement a persistent state store.


EdgeX Micro Service
Alert/ Notification Client registrations
On applicable change of state or condition notify via callback
Alert/Notification Registration API
register
Management Service Agent as Alert/ Notification Client
Other Alert/ Notification Clients

To begin with, each micro service should offer alerts/notifications on the following state or condition changes:

- Notify on change to a configuration setting (provide the key or name of the configuration setting to watch)

Commented [Tony Espy12]: Bullet formatting is still off...

Notify on change of the operating state between enabled and disabled states.

Notify on change of the admin state between locked and unlocked states.

Notify when the RAM memory usage falls outside of min/max range. Note: the accuracy of this information will be subject to the implementation technology of the micro service (example Java versus Go) and its capabilities. What is returned is expected to be the total memory used by the micro service (versus something like heap space).

Commented [Tony Espy13]: Still not convinced this should be the responsibility of the micro services vs. using native OS or deployment mechanisms.

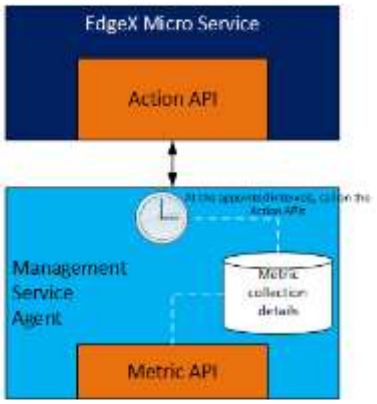
Additional state or condition changes may be offered via the Alert/Notification API in the future (such as heap space versus total memory usage).

Each of the above cases, the client must provide the client callback information (along with any other pertinent details such as the particular configuration setting of interest, or the min/max RAM range) when registering for the alert/notification.

Metric API

A metric is defined as a time varying value of the micro service that is sampled according to a particular interval. For example, get the amount of memory used by the micro service every 30 seconds.

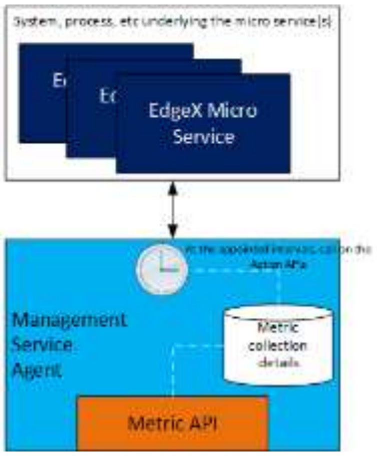
From the micro service perspective, metric collection is just another call to an Action API. It is the Agent that must manage the collection of the data on the appointed interval(s).


EdgeX Micro Service
Action API
At the appointed intervals, call on the Action APIs
Management Service Agent
Metric collection details
Metric API

The Agent must offer an API (outlined below) to allow for the creation, removal, and general management of the metric collection.

- Add a new metric (specify the micro service and Action API to call, the metric to capture, and the interval). Note: the collection time will be specified via CRON style string.
- Remove a metric (specify the micro service, and metric to be stopped).

In the future, some metric collection may not require communication with the micro service Action API. In some cases, the Agent may determine the metric based on knowledge of the system, OS, etc. For example, the request for a micro service's CPU usage may be requested from the operating system for the process that corresponds to the micro service.


System, process, etc underlying the micro service(s)
EdgeX Micro Service
At the appointed intervals, call on the Action APIs
Management Service Agent
Metric collection details
Metric API

Because of the dependency on elements outside of EdgeX Foundry micro service code, this type of metric collection is out of scope at this time.

Any metric data collection is meant to be idempotent with regard to the micro service – meaning metric data collection does not change the state of the micro service nor does it cause the micro service to do anything other than report on the value of the metric data.

In the first iteration of the Metric API, the Agent (and associated micro services) must be capable of supplying the following metrics:

- operating state (***enabled/disabled***),
- admin state (***locked/unlocked***)
- memory usage

Commented [Tony Espy14]: I still am not convinced these are valuable as metrics vs. alerts.

Future versions of this API may offer metric information on CPU load, storage input/outputs per second, communication input/output per second, number of REST error replies, number of REST request failures, number of REST requests. Future versions of this API may also consider adding metrics for specific micro services (or types of micro services). For example, device services may report on the number of device/sensor requests made, or amount of time a sensor was "offline."

Management Service Agent

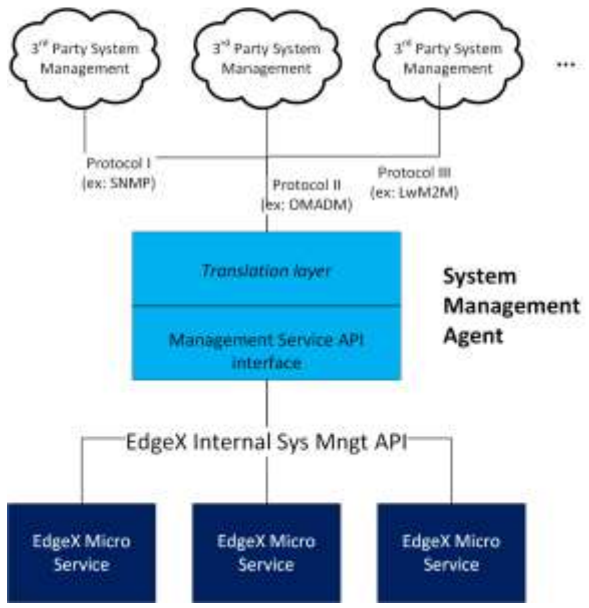
The management service agent (aka Agent) translates 3rd party system management requests to EdgeX management service API calls (see below) on EdgeX micro services. For example, it would take an LwM2M request to check on the health of a micro service and translate that to the EdgeX ping action.

The Agent will also multiplex requests when necessary. For example, a request to "shutdown" EdgeX from a 3rd party management system would result in requests to stop each micro service.

As an initial implementation, the Agent will simply expose the EdgeX management service API to 3rd party systems (without translation). In other words, the Agent serves as a proxy for system management service API calls into each micro service.

System management API calls made to the management agent must specify an additional parameter (name) to identify the target micro service, unless the operation applies to all micro service (eg. a full shutdown request). In the later case, the system call would be translated to individual micro service API calls by the agent.

Time permitting, a single translation to an alternate system management API (i.e. OMA LwM2M, DMTF/Redfish, OMA DM, TR69, DOCSIS, SNMP, etc.) would be provided.

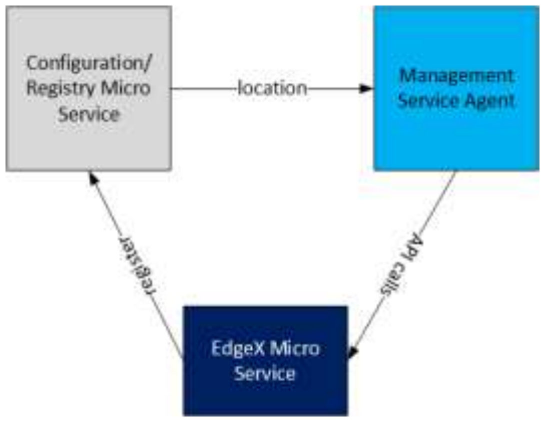


The Agent will work with the assistance of the Configuration/Registry micro service, to call on any of the management service API. By default, when EdgeX is operating as intended, the Configuration/Registry micro service will know the location (REST IP address and port) and availability of each micro service. This is because all micro services must initially register themselves with the Configuration/Registry service when they start.

Given that the Configuration/Registry service knows the location of each micro service, the Agent will need to request (by name) the location of any micro service through the Configuration/Registry micro service when there is a need to call on a micro service's management service API. The Agent can cache the location of each micro service but then must avail itself to be notified when the micro service is no longer available at its cached location.

When caching the location of each micro service, the Agent shall provide the Configuration/Registry micro service with a callback REST endpoint address that the Configuration/Registry must call whenever a micro service location or operational status changes.

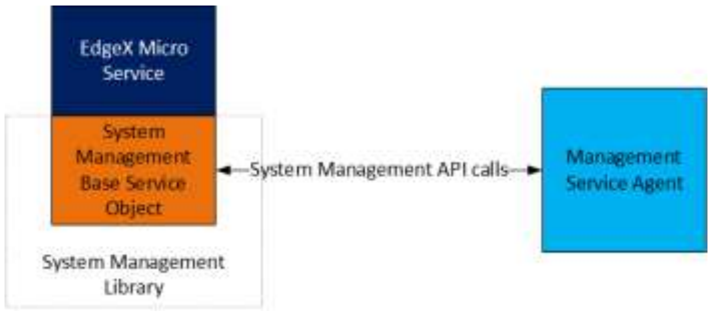
By default, and when EdgeX is operating in good order, the Agent will call on the Configuration/Registry micro service to get micro service location and status as described above. But, in circumstances where the Configuration/Registry micro service is not available, the Agent will operate first on any cache information it has, and second by a set of default locations for the micro services provided via property file resource stored with the Agent program artifact. This behavior mirrors the use of local program artifact application.properties in the existing micro services in the event that Consul is not available to provide configuration to the micro services.



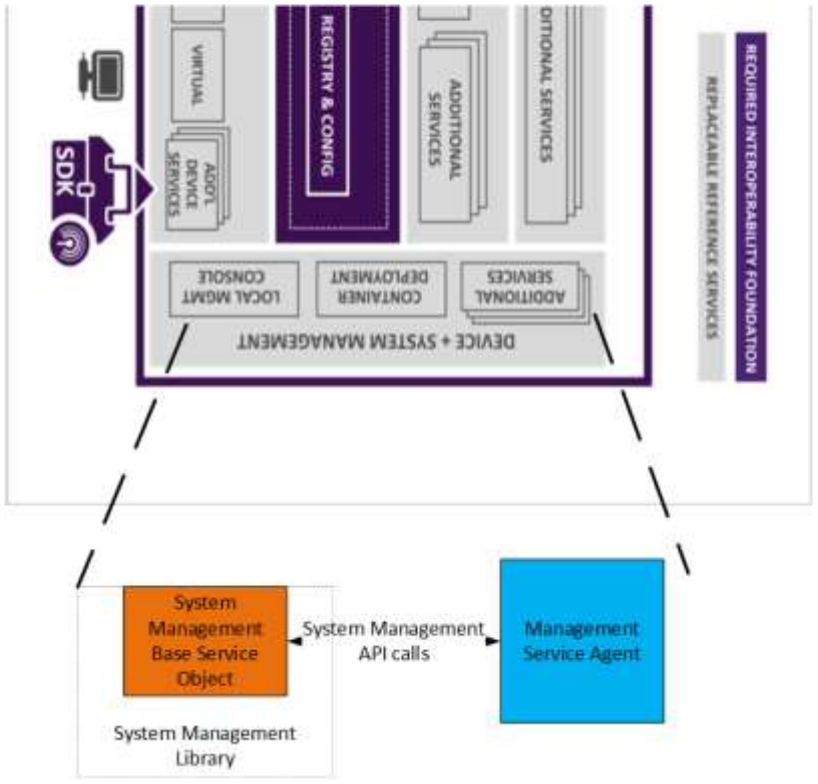
Management Service API Implementation

EdgeX believes in polyglot development. Therefore, each micro service can be implemented with different technology. In order to satisfy the needs of the EdgeX management service API, a reusable library of system management functionality should be created (and shared) in the various languages used in EdgeX micro services.

In Go, this will mean a package of shared functions to be made part of each Go micro service. In object oriented languages like Java, it is envisioned that the system management library will offer a base micro service interface and corresponding base class that implements the above management service API.


EdgeX Micro Service
System Management Base Service Object
System Management API calls
Management Service Agent
System Management Library

The Agent and Management Service API will make up the initial “System Management” functionality depicted in the current EdgeX architectural diagram.


VIRTUAL
ADD'L DEVICE SERVICES
SDK
REGISTRY & CONFIG
ADDITIONAL SERVICES
DITIONAL SERVICES
REQUIRED INTEROPERABILITY FOUNDATION
REPLACEABLE REFERENCE SERVICES
LOCAL MGMT CONSOLE
CONTAINER DEPLOYMENT
ADDITIONAL SERVICES
DEVICE + SYSTEM MANAGEMENT
System Management Base Service Object
System Management API calls
Management Service Agent
System Management Library

Not in Scope for the Micro Service System Management

The following functionality is outside of the scope of the California release.

- Software updates.
- The Managed Object that corresponds to the underlying system and to connected devices.