System Management, Phase I (Delhi Release), Design

Version 3, 7/24/18

Microservice SM Agent (SMA)

Design Considerations

SMA is an optional service

We have to accept that the system management agent (and some of the API in the sys management API of each service) may be provided by more industrial / enterprise capability like Kubernetes or cloud deployment/orchestration facilities. So we have to have a platform that allows the agent/apis to be turned off.

Each service containing the SM API must have a configuration setting that turns off, protects or otherwise causes the SM API on the services to no-op. This makes sure some rouge process does not use the API and bypass some other system management capability.

How to determine "all" services

The system management agent (SMA) must know what services it manages and where each service is located. It often makes a call to all of the EdgeX microservices (ex: to stop all services or to start all services). How is "all services" defined or determined? When Consul is running, it can provide a catalog of registered services, but this exposes a chicken and egg problem. If the SMA is requested to restart all services, for example, it could use Consul information to get all services and issue a stop command to each, but how would it be able to then send a start command to each if Consul is not even up yet?

Further, EdgeX has been created to operate without a registry/config service – especially for developer environments. Can the SM Agent operate without config/registry?

In order to facilitate the independent operation of SM and one that may bootstrap all of EdgeX, the SMA will be provided configuration (a manifest) that specifies the services it is to manage. As with all services today, a bootstrap property (like –consul) will indicate to the SMA to get the configuration from the local file system or from Consul. The config will list the services that the SMA is managing and provide information on start/stop of each. More details on this configuration file is provided below under SMA Manifest.

SMA Manifest - Service Registry and Start/Stop Commands

As the SMA must be able to start and stop each service (restart too but this is just a combination of stop and start), it must know how to issue the appropriate command for start and stop of each service. Unfortunately, the start and stop of each service may be directly associated to containerization (Docker start/stop commands), service technology (Java, Go, etc.) and many other factors.

While EdgeX can attempt to standardize how its services are managed, there are associated 3rd party infrastructure services (MongoDB, Consul, Kong, etc.) that will not adhere to EdgeX guidelines. In fact, these services will have to be managed differently in that they will also not be able to respond to the SM API set.

A manifest configuration file must be provided to the SMA (either via Consul or the local file system as indicated by a bootstrap property like –consul). This file must contain the names of the services, the intended location of the services (which for now would always be on the same host), and the commands to issue in order to start/stop the service. The manifest will also signify where the service is an EdgeX microservice (and conforms to the SM API) or if the service does not (MongoDB, Consul, etc.) and therefore not be subject to standard EdgeX API calls.

This file will boostrap the SMA. Different versions of the file may exist depending on how/where EdgeX is deployed (Docker v. Snappy, Windows v. Linux, etc.).

In this first release of the SMA, the manifest file will be static. In the future, the manifest may be more dynamically contrived or provided by some 3rd party orchestrator.

System of record for configuration

While Consul is generally regarded as the home for configuration information, each service gets the configuration from either Consul or its local configuration file depending on startup parameters. So, in reality, only each service knows what its real configuration. Therefore, the SMA must request configuration information from the service itself - and allow the configuration to come from Consul or a local configuration file as deemed by that service.

Metrics Significant Change

On any dynamic metric, like memory usage, clients that have interest in the change are usually interested in "significant" change. But the significance may be environmentally or client dependent. Therefore, on startup of any service, a configuration property must be specified that indicates the min/max range for the metric. The range can be different per service and per metric. Registered clients are notified by the service when the metric is outside the range.

SM Agent API

The SMA must be able to respond to the following public REST APIs

- Stop all EdgeX microservices
    - api/v1/stopall
- Stop an EdgeX microservice by name
    - api/v1/stop/**servicename**
- Start all EdgeX microservices
    - api/v1/startall
- Start an EdgeX microservice by name
    - api/v1/start/**servicename**
- Restart all EdgeX microservices
    - api/v1/restartall
- Restart an EdgeX microservice by name
    - api/v1/restart/**servicename**
    - predicated on no – sleep time between stop/start
- Ping an EdgeX microservice (check that the service is still up)
    - api/v1/servicename/ping
- Get the configuration settings (aka properties) for an EdgeX microservice by name
    - api/v1/config/servicename
    - Hateos pattern – consider and include along with Emad
    - https://spring.io/understanding/HATEOAS

- Get the configuration setting (aka property) for an EdgeX microservice (by name) by configuration setting name (aka key)
    - api/v1/config/**servicename/key**
- ~~Set the configuration setting (aka property) for a writable (versus read only) property for an EdgeX microservice (by name). Example: the port of a service is read only whereas the log level maybe updated (writable).~~
    - ~~api/v1/config/**servicename/key** [POST] with new value as body~~

      Removed for the release due to complexity of issues and shortened delivery window
- Get the memory usage for an EdgeX microservice by name
    - api/v1/memory/**servicename**
    - **Need full API with JSON**
    - **May need additional params (like force stop on start/stop commands)**
- ~~Get the current Admin status (locked or unlocked) of an EdgeX microservice (only valid for Device Services?)~~
    - ~~api/v1/adminstatus/**servicename**~~
    - *Removed for the release due to complexity of issues and shortened delivery window*

- Register/deregister a client for change to a configuration setting, status change, or significant memory usage change to a named EdgeX microservice
    - api/v1/register/**servicename** [POST / DELETE] with the client callback and element of interest (config, metric, admin status) specified in the body

An SMA package defines these functions:

Stop()

For each service, use the local manifest (likely cached on bootrstrap) to issue the appropriate stop command to each service.
Issues:

- Does each service need to acknowledge back? Are there dependencies which require acknowledgement before stopping others?
- *discussion*
- *Wouldn't want caller to do all that magic – return a token from SMA and use that token to check on status from SMA*
- *Would require SMA to have state management*
- *Crawl before walk would say do it synchronously*
- *Walk to run would be async later*
- *Sync call to stop,*
    - *Want to async send stop to each service*
        - *Whole operation gets a timeout – reports what acknowledged and what did not acknowledge by timeout*
    - *SMA then responds 200 & list of acknowledged vs non-acknowledged*
    - *SMA does not stop itself (but SMA needs its own stop API)*
- *Noop-handling for now for checking if stop really did happen*
    - *Future – determine how / what actually stopped – but differs per language, OS, docker, snap, etc.*

- *Should we have a look at other system management APIs – MEC, OpenStack/Windriver (StarlingX), Kubernetes, Docker, sysd, Snappy*

Stop(servicename)

Stop the service using the stop command specified in the local manifest.
Issues:

- How to respond if servicename does not exist?
- What if other services are dependent on the named service?

Start()

For each service, use the local manifest (likely cached on bootrstrap) to issue the appropriate start command to each service.
Issues

- Does order matter? Presumably it does. If so, what happens on start if one service cannot be started?
- Handle by manifest order
- Support it "in case" – but we are working to ameliorate so we don't have time dependencies. But we don't want another place where configuration and order of that kills. Therefore – services must be decoupled from tight dependencies.

Start(servicename)

Start the service using the start command specicfied in the local manifest.
Issues:

- How to respond if servicename does not exist?

Restart()

For each service, use the local manifest to stop and then start each service (with same issues applying from above)

Restart(servicename)

Stop and restart the service using the stop and start command from the local manifest (with same issues applying from above)

Ping(servicename) – *drop because we are duplicating functionality; also it may come from Docker or Snappy, etc.*

SMA may eventually need to report best it can what is running (see start/stop functionality), but this is dependent on system, OS, language, etc.

If the service is an EdgeX service (versus MongoDB, etc.), then call on the ping operation of the service.
Issues:

- Service is not up or available, what's the default response?

Config()

For each service that is an EdgeX service, collect and return the configuration for all services by calling on each the Config operation of each.
Issues:

- What if a service is not up to respond?
- What is the structure of the return given multiple services and multiple configs?
- *Refactored structure part of Config-v2 – JSON*
    - *TOML is file format and easier to write for human. Internally, everything is JSON object*
    - *Side issue – Device Service and Core WG needs to align on config-v2*

Config(servicename)

This operation should only apply to EdgeX services (versus MongoDB, etc.). Collect and return the configuration for the specified service by calling on the Config operation of the service.

Issues:

- What if the service is not up to respond?
- What is the structure of the return? It should be consistent with Config() call.

~~UpdateConfig(servicename, key)~~

~~This operation should only apply to EdgeX services (versus MongoDB, etc.). Call the UpdateConfig function on the service to update the value in the configuration structure.~~

~~***Issues*:**~~

- ~~What if the service is not up to respond?~~
- ~~In order for this to remain persistent, it would have to change the value in Consul or in the local properties file. Do we allow this?~~
- ~~How can we determine if a property is writable?~~
- ~~How to receive and acknowledge the update?~~

Memory(servicename)

This operation should only apply to EdgeX services (versus MongoDB, etc.). Call the Memory function on the service

Issues:

- As there will probably be many metrics requests in the future, should this be a generic Metric(servicename, metricname) to allow for more dynamic capture in the future? Implementation of each type metric capture could be very different under the covers.
- What if the service is not up to respond?
- Exactly what to report and in what unit of measure from the stats

~~AdminStatus(servicename)~~

~~This operation should only apply to EdgeX services (versus MongoDB, etc.). Call the AdminStatus function on the service~~

~~***Issues*:**~~

- ~~What if the service is not up to respond?~~

Register(servicename)

Depending on POST or DELETE operation, register or deregister for changes in a service (either config, admin status or memory usage)

Issues:

- Who/what keeps the registration information persistent?

- How to specify a client and what protocols are supported? Probably best to first start out with just HTTP clients – specifying an endpoint URL to call when event change has occurred
- How to specify the registration of interest (config change, admin status change, memory change)
- Format of data to send on any change in the service

Gorilla Mux or other router, directs the REST client requests to these methods.

Microservice System Management (SM) API

Each EdgeX micro service must be able to provide the following public REST APIs

- Stop this microservice
  - api/v1/stop
- ~~Get the current Admin status (locked or unlocked)~~
  - ~~api/v1/adminstatus~~

  Removed for the release due to complexity of issues and shortened delivery window

- Get the configuration settings (aka properties) for this microservice
  - api/v1/config
- Get the configuration setting (aka property) for this microservice by configuration setting name (aka key)
  - api/v1/config/***key***
- ~~Set the configuration setting (aka property) for a writable (versus read only) property for this microservice~~
  - ~~api/v1/config/**key** [POST] with new value as body~~

  Removed for the release due to complexity of issues and shortened delivery window

- Get the memory usage for this microservice
  - api/v1/memory

A system management package defines these functions (which could also be interfaced like db.go). Ideally, the package is common and used by all services in the same way.

Stop()

Initially, this may be a simple call to os.Exit().

Issues:

- Eventually need a graceful shutdown. Options to use context (https://medium.com/@matryer/make-ctrl-c-cancel-the-context-context-bd006a8ad6ff) and/or something like this: http://guzalexander.com/2017/05/31/gracefully-exit-server-in-go.html for more graceful exits.
- How does this impact the service that is dockerized (or otherwise containerized)? By killing the executable will Docker detect and also shutdown?

AdminState()

Today, not all microservices have or use an AdminState (something to consider for future releases). Only device services have an adminstate which is obtained via call to metadata. Check that it is a device service and if so, return adminstate which should be same as what is in metadata.

Issues:

- how to know if a service is a device service or not?
- What to return on non-device services if this functionality is added generically to all services by package

Config()

Per core and support services, each service loads configuration from the file system or Consul at startup (under Init.go). A call to this function can simply map and return the configuration structure to simple key=value pairs already loaded by the service.

ConfigFor(key string)

Per core and support services, each service loads configuration from the file system or Consul at startup (under Init.go). A call to this function returns the value at the key for the config structure.

Issues:

- Need to address issue of what to return when the key is not found.

~~UpdateConfig(key string, value string)~~

~~Per core and support services, each service loads configuration from the file system or Consul at startup (under Init.go). A call to this function will update the value in the configuration structure.~~

~~***Issues:***~~

- ~~In order for this to remain persistent, it would have to change the value in Consul or in the local properties file. Do we allow this?~~
- ~~How can we determine if a property is writable?~~

Memory()

Create a function that returns data from Go Lang's memory stats. See https://golangcode.com/print-the-current-memory-usage/ and https://golang.org/pkg/runtime/#MemStats.

May want to capture metrics to a persistent store/cache.

May want to provide an abstraction layer for implementation of the collection of data (or any of these functions) so that it allows for different implemenations by 3rd parties or even EdgeX in the future. For example, the 3rd party may provide the version that takes care of persistence or pushes the metric data to some place like Grafana.

Issues:

- As there will probably be many metrics requests in the future, should this be a generic Metric(metricname) to allow for more dynamic capture in the future? Implementation of each type metric capture could be very different under the covers.
- Exactly what to report and in what unit of measure from the stats

Gorilla Mux or other router, directs the REST client requests to these methods.

Microservice System Management (SM) Callbacks

Each EdgeX micro service must be able to provide the ability to inform interested clients of changes to its configuration, admin status and memory usage (potentially other metrics in the future).

On any configuration or admin status change, the service should request a list of interested clients from the SMA. Each interested client's callback address should then be invoked with details on the change.

Memory usage (or other resource check) will require each service have a timer that calls to check on its memory (or other resource in the future) on a routine / configurable basis. With each check, the value will be compared to a service configuration specified range (min and max value). When the value of the check is outside of the range, each interested client's callback address should then be invoked with details of the change.