EdgeXFoundry Configuration Management (Proposal v3)

The context for this document is to describe how configuration data supporting the EdgeXFoundry platform should be organized, managed and interacted with. The focus is primarily on the external provider of configuration data, such as Consul, used during Dockerized or cloud-based deployments. Local or native deployments simply use the configuration.toml files included with each service.

The primary aspects under discussion are as follows:

- Top-level reorganization
- Readable vs Writable settings
- Versioning

The information presented below takes the view of the upcoming Edinburgh release as needing long term support (LTS). It also assumes that legacy strategies or data (such as *.properties files supporting old Java device services, or current paths like config/V2) can be eliminated from consideration and do not constrain the strategy going forward.

Top-level Reorganization

Configuration information should be organized into a hierarchical structure allowing for a logical grouping of services, as well as versioning. The current practice whereby all service configurations are children of a /config/v2 parent will be eliminated.

Going forward, the proposal is to provide for a logical grouping of services beneath a root "edgex" namespace at the very top level of the configuration tree. The top-level namespace separates EdgeXFoundry-related configuration information from other applications that may be using the same registry. Below the root, sub-nodes will facilitate grouping of device services, EdgeX core services, possibly security services, etc. For these examples, the top-level nodes shown when one views the configuration registry would be as follows:

- edgex *(root namespace)*
  - core *(edgex core services)*
  - devices *(device services)*
  - security *(security services)*

This enables independent management of the configuration solution by service implementations currently residing in separate repos, allowing the possibility of different release cadences. It provides a clear delineation for where the configuration of a given service type should reside as opposed to having all things in one big tree and further, removes the need to have one service with knowledge of how to seed all of this information into the registry (which implies release coupling between separate repos).

Versioning

An example of the current recommendation is shown below.

- edgex
  - core *(edgex core services)*

        - 1.0
            - Edgex-core-command
            - Edgex-core-data
            - Edgex-core-metadata
        - 2.0
    - devices *(device services)*
        - 1.0
            - mqtt-c
            - mqtt-go
            - modbus-go
        - 2.0

The versions shown correspond to major versions of the given services. These are not necessarily equated with long term support (LTS) releases. For all minor/patch versions associated with a major version, the respective service keys would live under the major version in configuration (such as 1.0). Changes to the configuration structure that may be required during the associated minor version development cycles can only be additive. That is, key names cannot be removed or changed once set in a major version, nor can sections of the configuration tree be moved from one place to another. In this way backward compatibility for the lifetime of the major version is maintained.

An advantage of grouping all minor/patch versions under a major version involves end-user configuration changes that need to be persisted during an upgrade. The config-seed will not overwrite existing keys when it runs unless explicitly told to do so. Therefore if a user leaves their configuration registry (Consul) running during an EdgeXFoundry upgrade, only the new keys required to support the point release will be added to their configuration, leaving any customizations in place. If we were to instead have an entry in the above configuration tree for every minor version, with the whole configuration tree underneath, then the community thinks we would be obligated to write a migration tool to bring over any customized settings. This proposal seeks to eliminate the planning and implementation overhead involved in building such a tool.

Readable vs Writable settings

Within a given service, there will be keys whose values can be edited and change the behavior of the service while it is running versus those that should are effectively read-only. After discussion with the community, the direction for grouping writable settings is to place them within their own group under a service key, similar to how we group other settings. For example, the top-level groupings for edgex-core-data would be:

/edgex/core/1.0/edgex-core-data/Clients
/edgex/core/1.0/edgex-core-data/Databases
/edgex/core /1.0/edgex-core-data/Logging
/edgex/core/1.0/edgex-core-data/MessageQueue
/edgex/core/1.0/edgex-core-data/Registry
/edgex/core/1.0/edgex-core-data/Service
/edgex/core/1.0/edgex-core-data/Writable

There are three levels of configuration that have been identified.

1. Read-only, never change
    a. These are settings which are not editable in any fashion by a user or outside application.
    b. In actuality, these settings would most likely never be found anywhere but in the code itself, for example as a literal or constant.
2. Read-only, change on restart
    a. Changes to these settings do not affect currently running services until restart.
3. Writable
    a. These are settings found in the Writable section.
    b. Changes to these settings should immediately affect the behavior of a service without a restart.

A question was raised as to whether or not each of the above sections should have its own Readable/Writable grouping. An example use case is a user who might wish to change the level of logging output in real-time while diagnosing a problem. They might want logs to emit only ERROR level message by default but while troubleshooting, change that to DEBUG. I suggest that we not interleave Writable sections into each configuration section within a service because that will increase the number of Registry watchers we need to utilize. In this case, then, the setting for LogLevel would be moved out of Logging and into Writable. The service would then only need to watch one path in its configuration tree *(e.g. /edgex/core/1.0/edgex-core-data/Writable)*