Geneva

The Geneva Release is the 6th successful community release of EdgeX Foundry. It is a minor (dot) release (version 1.2) and therefore backward compatible with the Edinburgh (v1.0) and Fuji (1.1) releases. Although a minor release, a number of new and significant features and improvements are unveiled with the release to include automatic device provisioning, a new rules engine implementation, and batch and forward functionality to name a few.

Release Information

- Dynamic / automatic device provisioning/on-boarding
- Alternate messaging support (RedisStreams, MQTT, 0MQ, …)
- Better type info in sensor data collection
- REST device service
- Batch and send
- Use of secrets for authenticated MQTT/HTTP exports
- Sensor collection of an array of types
- Redis as the default DB
- New rules engine (Kuiper vs Drools – written in Go; smaller / faster)
- Improved Security
- Interoperability testing
- DevOps Jenkins Pipelines

Adopter Warnings

- The EdgeX community has deprecated three features in this release of EdgeX. Deprecation does not mean the feature is removed from this release, but it is a strong indication that the feature will be removed in an upcoming release.
    - MongoDB support has been deprecated. With the Fuji release, the community decided to make Redis our default database. We now have intentions of removing MongoDB from EdgeX in the near future. All code associated to storing/retrieving EdgeX data in Mongo has been labeled as deprecated. Docker Compose files for MongoDB include a warning to let you know support for MongoDB is coming to an end. Redis has been selected over MongoDB by the community for ARM support, memory/footprint improvements, license considerations and because of the involvement of the Redis Labs in the project.
    - The Support Logging service has been deprecated. The community felt that there are better log aggregation services available in the open source community or by deployment/orchestration tools. In this release, the logging service is not started with the EdgeX provided Docker Compose files (it is still in the file but commented out). By default, all services now log to standard out. If users wish to still use the central logging service, they must configure each service to use it. Users can still alternately choose to have the services log to a file.
    - The Support Rules Engine (which wrapped the Java-based Drools rules engine) has also been deprecated. EdgeX is partnering with the Kuiper project to provide new and proved rules engine support.
- Redis is an open source (BSD licensed), in-memory data structure store, used as a database and message broker in EdgeX. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams. Redis is durable and uses persistence only for recovering state; the only data Redis operates on is in-memory.
    - Redis uses a number of techniques to optimize memory utilization. Antirez and Redis Labs have written a number of articles on the underlying details (http://antirez.com/news/92, https://redislabs.com/blog/redis-ram-ramifications-part-i/, https://redis.io/topics/memory-optimization) and those strategies has continued to evolve (http://antirez.com/news/128). When thinking about your system architecture, consider how long data will be living at the edge and consuming memory (physical or physical + virtual).
    - Redis supports a number of different levels of on-disk persistence. By default, snapshots of the data are persisted every 60 seconds or after 1000 keys have changed. Beyond increasing the frequency of snapshots, append only files that log every database write are also supported. See https://redis.io/topics/persistence for a detailed discussion on how to balance the options.
    - Redis supports setting a memory usage limit and a policy on what to do if memory cannot be allocated for a write. See the MEMORY MANAGEMENT section of https://raw.githubusercontent.com/antirez/redis/5.0/redis.conf for the configuration options. Since EdgeX and Redis do not currently communicate on data evictions, you will need to use the EdgeX scheduler to control memory usage rather than a Redis eviction policy.

Known Bugs

- Sensor readings are captured event/reading objects by device services. The event/reading objects are used throughout EdgeX to transport the sensor data. An event can have multiple readings associated. For example, in a single sensing, a thermostat may create an event with two readings: one for temperature and one for humidity. There are currently no restrictions on the number of readings on an event. In theory, an event can be packed with so many readings that it creates a object that is MBs in size or even larger. This can cause the system, particularly core data, start to slow down or even fail. In a future release, appropriate governors will be put in place to prevent event/reading objects from getting too big. For now, users are advised to monitor the size of their event/reading objects and break them apart if there are too many or too big of readings associated to an event. Ref #2527.
- Removing a scheduler interval by ID when an interval action is still using it crashes the scheduler. Ref #2520. The underlying problem is that the REST call to delete the scheduler interval by ID does not perform a check to ascertain that the interval is currently in use. As a work around, users are advised to follow the following procedure:
    - At the outset of interacting with the support-scheduler service, first use the REST API to get the entire interval information. For example, let's say that the ID happens to have the value 71e5c882-61b8-4d98-a3b7-2aa68d60c5e6, then issue this cURL command (using the ID value get the full interval information: curl --location --request GET 'http://localhost:48085/api/v1/interval/71e5c882-61b8-4d98-a3b7-2aa68d60c5e6'
    - From the response received, make a note of the interval's name and use the support-scheduler API to delete the scheduler interval by name (versus ID). If the name of the scheduler interval was midnight, the call would look like the following cURL command: curl --location --request DELETE 'http://localhost:48085/api/v1/interval/name/midnight
- The services had a set of configuration (in configuration.toml - see properties below) meant to configure how a service attempted to try to connect to a database upon initial connection failure. A duration configuration indicated how long to wait before the next retry and interval indicated the number of retries before abandoning the connection setup and exiting the service. This configuration did not work. A fix for this has been created for Hanoi (https://github.com/edgexfoundry/go-mod-bootstrap/pull/86), but these properties will not cause any effect today. Internal to the services, there was a default retry mechanism in place and that will still take place today in the event that a service cannot initially connect to the database. As of Geneva, a service will **still** attempts to connect to the Database approx every second and gives up after approx 30 seconds.
    - ```
      [Startup]
      Duration = 10
      Interval = 5
      ```

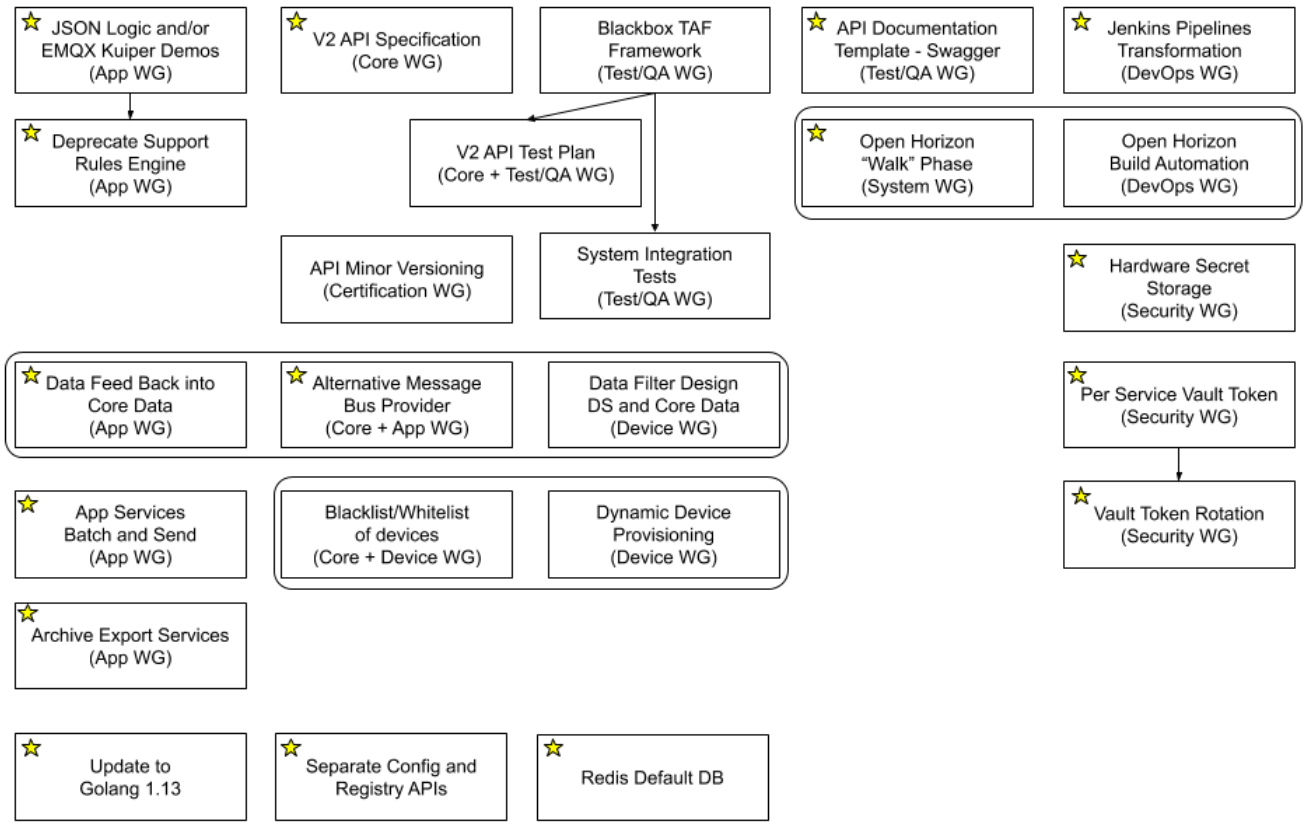
SDKs, Device and Application Services

Please note that device and application services (along with the associated SDKs) can and do release minor versions independently. These services and SDKs must maintain backward compatibility with EdgeX releases.

Check with the release notes for the SDKs for details on changes and issues addressed in each of the SDK minor releases.

- C Device Service SDK: https://github.com/edgexfoundry/device-sdk-c/blob/master/CHANGES
- Go Device Service SDK: https://github.com/edgexfoundry/device-sdk-go/blob/master/RELEASE-NOTES.txt
- App Functions SDK: https://github.com/edgexfoundry/app-functions-sdk-go/blob/master/CHANGELOG.md

Geneva Release Overview
      ```



Release Dates and Timeline

Code Freeze: Apr 22, 2020

Release: May 13, 2020