Geneva Release Notes

5/13/20

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 Major Themes

- 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 an 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. Keep an eye on the database growth and the number of events and readings in the database (the core data API reference gives you the means to count events and readings). Routinely run core data clean up functions to clear out pushed or remove old events/readings. 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

General

- New and updated Geneva docker-compose files have been created
    - Portainer was removed from the release EdgeX compose files and made part of a separate compose file for inclusion when needed but not by default
    - Compose files will indicate Mongo, Logging service and the Support Rules Engine are deprecated and to be removed in an upcoming release
    - Export services are now archived and are removed from the compose files.
- Documentation updates for the platform have made
    - Additionally, documentation was moved from RST to Markdown format
    - This allows for much more pleasant looking documentation along with more searchable documentation
- Move to Go 1.13
- Redis has been made the default reference implementation database for all service persistent storage needs
    - Mongo is still available for this release as an alternate implementation, but will be removed in an upcoming release
    - Mongo references are marked as deprecated in the code platform
- Pull Request (PR) templating has been setup for the project to improve comments on all PR submissions
- The project implemented an Architectural Decision Records (ADR) process and document template for exploring all architectural decisions in the project
    - ADRs are stored in the edgex-docs repository in Github (https://github.com/edgexfoundry/edgex-docs/tree/master/docs/1.2/design/adr)
- Removal of export services
    - The EdgeX Export Client and Export Distro micro services have been archived and removed from the Docker Compose files
    - These services were deprecated with the Fuji release. Export services are replaced by Application Services
- A common service bootstrapping module (go-mod-bootstrap) was developed and used in all services (and SDKs)
    - This bootstrapping provides a common way for all services to ingest configuration information

- The config-seed service was removed in favor of service self-seeding. (documented via ADR 0005)
    - This greatly simplifies service configuration and removes a service that only initialized and then exited (leading to confusion on the part of users)
- The Logging service, Support Rules Engine service and use of MongoDB have been deprecated in this release
    - While still available, all will be removed in a future release
    - The community felt that the central logging service no longer served a purpose as there was rarely a need to log to a database
    - There are often better logging aggregation tools available in the marketplace
    - Services can either log to a file or to standard out, which were the more often used logging options
    - The decision to use Redis in place of Mongo has been in the works for some time
    - Redis offers a reduced footprint, works in x86 as well as ARM environments with fewer issues, and offers a better license model for an Apache 2 project
    - The Support Rules Engine service was written in Java and was the last of the legacy EdgeX services. EdgeX is now partnering with the Kuiper rules engine project which is written in Go. This will make inclusion of a rules engine must smaller, faster and lighter – plus Kuiper rules can be specified in SQL.

Core & Supporting Services

- The platform now supports passing an array of types (example, an array of integers) in EdgeX readings from Device Services to Core Data and beyond
    - This allows the collection of sensor data to maintain sensor values in a cohesive and small object unit versus creating multiple readings to do the same work
    - Support for the use of an array of integers in driving rules engines (Drools, Kuiper, JSONLogic) has been deferred to the next release
- The platform has been updated to allow for secure Redis deployment (where credentials are required by the platform or 3rd parties to access the database)
- The Drools rules engine – the last remaining Java service in EdgeX – has been deprecated (see note on Kuiper below)
    - For this release, as Drools can still be used, the service has been updated to allow support-rules-engine (Drools) to always use local config since the config-seed has been removed
- With automatic device provisioning added with this release (see Device Services below), backlisting support was added to metadata to allow a device service to exclude onboarding some devices automatically
- Reading type was added to Readings (via go-mod-core-contracts)
    - This allows applications to have better understanding of sensor data without making a request of Metadata to get the value descriptor
    - Specifically: “valueType” was added while “mediaType” and “floatEncoding” were added as optional properties to be included for floating point / binary data as appropriate

- Separate registration and configuration modules were created so that these services could be implemented by different implementations in the future and to allow for better separation of concerns (see go-mod-registry & go-mod-configuration)
- MQTT & Redis Streams implementations of the message bus abstraction have been added in go-mod-messaging
    - This primarily benefits communications between core data and Application Services today where messaging is used between the services, but the messaging module allows for future messaging with varying implementations to be used by any service in the future
    - The EdgeX reference implementation for core to app service communications still uses 0MQ, but example MQTT and Redis Streams implementation have been documented
- A number of superfluous and unused properties on the value descriptor were removed (uomLabel, formatting)

Device Services (and SDK)

- Changes associated with Config Seed removal & service self-seeding and support for array of types as listed under Core
- Also added type to Readings as specified above in Core
- Implementation of dynamic device provisioning (i.e. automatic provisioning) in the Device Service SDKs (and associated device services) to allow device services to automatically add a device under management
    - The implementation of dynamic device provisioning will vary per protocol/device type
    - The SDKs provide the common boilerplate code that allows DS implementers to add this feature more directly/quickly
- Released version 1.2.0 of SDK API in C (including a number of internal API changes)
- Released version 1.2.1 of SDK Go
- Provide the following new device services:
    - REST Device Service
    - Camera (for ONVIF cameras) Device Service
    - Device Service BACnet in C
- Improvements were made to Device Services to address consistency in endian-ness when generating float Readings (in Go v. C SDKs)
- Makefiles for C device services were added
    - This makes builds more consistent between Go and C device services

Application Services (and SDK)

- Released version 1.1.1 of Application Functions SDK
- The project worked with the EMQX Kuiper rules engine project (open source) to provide a long-awaited replacement to Drools
    - Kuiper is a Go based rules engine using SQL to help define rules
    - Use of Kuiper was added to Docker Compose files and integrated with the the App Service Configurable
- A JSONLogic function was added to the App Functions SDK and into the App Service Configurable

    - JSONLogic provides a “poor mans” rules engine that allows operations to be triggered based on pattern matches in the Event/Reading JSON data
    - An example of JSONLogic use was added to app-services-examples (in edgexfoundry-holding)
- Batch and Send functionality was added to the App Functions SDK and integrated into the App Service Configurable
    - This functionality allows EdgeX to collect sensor data readings and send them to a northside endpoint at designated times (versus as they arrive)
    - This allows EdgeX to be more easily used in environments where connectivity is intermittent or subject to failures
- The App Functions SDK and existing Application Services were updated to integrate the new go-mod-bootstrap
- The App Functions SDK and Application Services were updated to get and use secrets from Vault
    - This functionality was critical to allow authenticated communications
    - Per notes in Core, App Functions SDK and the Application Services were updated to use the new go-mod-messaging abstraction
    - This allows for 0MQ, RedisStreams or MQTT for message communications between core data and Application Services
- New App Service Configurable profiles were added for using MQTT/RedisStreams alternatives with the rules engine profile
- Added Insecure Secrets support
    - Allows application services to run without Vault, but still using authenticated MQTT/HTTP exports
- Added Exclusive Secret Store support for storing/retrieving app service specific (not shared) secrets
- Added REST API to push exclusive secrets (specific to the application service instance) into Vault
- Updated App Functions SDK (and Application Services) to use the additional type information is Readings from Core
- Provided new examples in edgexfoundry-holding repository for CloudEvent transformation functions

System Management (and CLI)

- Blackbox tests were added to cover System Management Agent (SMA)
- Metrics and operation requests now execute concurrently
    - That is requesting metrics or start/stop/restart operations against multiple services are run in parallel were previously they ran sequentially

Security

- Improved security service bootstrapping
- Improved security testing (Blackbox tests of APIs through the API gateway)
- Security Issue Review (SIR) team meeting regularly and addressing critical security issues (especially those identified by newly implemented Snyk reporting with this release (see DevOps below)
- All services now get their service secrets from Vault

- Created a token provider service
    - This service creates a unique Vault token for each individual service when it needs secrets from Vault
- Set up expiration/revocation/rotation of all tokens used by EdgeX
- Upgrade to Kong 2.0

Dev Ops

- Move Jenkins build from JJB to Jenkins Pipelines
- Now uses Advanced Synk Reporting (CommunityBridge) to detect security issues and provide vulnerability reporting
    - Weekly reports now being generated from this tool and reviewed by the SIR team
- Semantic versioning is now used/implemented in build pipelines
- Improved use of tooling and procedures to include use of GitHub Project Tracker, automated GitHub Issue label creation, and use of gitcommit linter
- New life cycle policies were implemented on Linux Foundation Nexus repositories.
- Developer documentation is now created via Pipelines
- Significantly improved performance of builds
    - Specifically got ARM builds down to lower than 15 minutes
- Implemented use of Snap Global Library
- Established release automation policy and procedures (ADR 0007)

Test / QA (and documentation)

- Implemented the first of backward compatibility testing
    - This will allow minor (dot) releases to check for any non-backward compatible changes in the future
- Started use of TAF tests to replace Postman / Neuman testing in the future.
    - This work included setup of TAF testing architecture and framework and TAF test code organization/repos
- Began EdgeX integration testing
    - This testing will be significantly increased in the future
    - The first integration tests check the correct flow of data from device service sensor data collection through to Application Services export
- Blackbox testing collaborative work as added for Application Services and Device Services (previously these services did not have blackbox tests)
- As part of EdgeX's collaborative work with fellow LF Edge Project Akraino, EdgeX blackbox tests were successfully run on Akraino/ELIOT blueprint and validated in the Akraino lab environment
- EdgeX documentation moved from RST to Markdown
- The EdgeX documentation site has been updated and enabled with Google Analytics
- EdgeX moved API documentation to Swagger documentation (OpenAPI 3.0)
    - API Documents are now available on Swagger Hub

EdgeX UI

- The EdgeX project combined and united its two UIs into a single reference user interface
    - As part of this work, the UI was significantly cleaned up and improved

- The new UI includes
    - Tools to use and configure the App Service Configurable
    - Improved interface to add, update, remove schedules (for the Scheduler Service)
    - Removed all tools using the now archived Export Services

Vertical Solutions

- Held a hackathon in China with more than 130 registrants
- Conducted the first EdgeX virtual hackathon with the help of Topcoder
    - This event had over 80 registrants and 20 projects competing

Open Horizons

- As a subproject under EdgeX, OH had its initial code contribution completed
- It successfully achieved LF Edge Stage 1 acceptance and is now a separate project
- Is in the process of developing its own automated build pipelines and testing utilizing EdgeX DevOps assistance

List of supported connectors

EdgeX Foundry application services (north side connectors) and device services (south side connectors) release on an independent schedule – although they must be compatible with the general EdgeX release. The list of device and application services below will work with Geneva (either currently or at the suggested timeline).

South side (Device Services)

Open Source Device Services

- Modbus (TCP/RTU) in Go
- Virtual Device in Go
- SNMP in Go
- MQTT in Go
- BACnet (IP & MSTP) in Go
- ONVIF Cameras in Go
- REST in Go (NEW with Geneva!)
- Bluetooth in C (summer 2020)
- BACnet (IP & MSTP) in C (summer 2020)
- OPC-UA in C (summer 2020)
- Grove C (summer 2020)

Commercially Available Device Services through community members

- File Exporter in C
- BLE in C
- Zigbee in C
- GPS in C
- CAN in C
- MEMS in C
- EtherCat in C
- Profinet in C
- EtherNet/IP in C (summer 2020)
- CanOpen in C (summer 2020)
- OPC-UA Pub/Sub in C (summer 2020)
- ONVIF in C (summer 2020)

North side (Application Services)

Available functions (as part of the SDK) for application services

- AES Encryption
- GZIP Compression
- ZLIB Compression
- MarkAsPushed
- PushToCore (new and experimental)
- Device Name Filter

- Value Descriptor Filter
- Batch
- JSONLogic Filtering
- XML Conversion
- JSON Conversion
- MQTT(S) Export
- HTTP(S) POST Export

Example Open Source Application Service Connections

- Azure IoT Hub
- Amazon IoT Core
- IBM Watson IoT
- Cloud Event Transformation
- For a more complete list of examples see project documentation.