Backlog

This page represents the work backlog for EdgeX. Items listed on this page are not yet approved and are not yet targeted for a specific EdgeX release. They have been and continue to be discussed for future releases. It serves as a "parking lot" for additions, enhancements, and changes /refactoring to be continually reviewed and considered with each release planning session. The order of items listed is random and should not imply priority. Items in the **No Longer Considered** list, on the other hand, have been considered at one point and have now been ruled out of consideration for the foreseeable future.

Feature additions and enhancements

- Support for RISC-V/64 bit architecture (per Levski planning 5/22)
- Future tuning consideration: how to lower JSON overhead (per Levski planning 5/22)
- Provide for internationalization/localization of metadata information (per Levski planning 5/22). One requestor wanted device names in Vietnamese.
- Provide for a time series database option in place of Redis for EdgeX (per Levski planning 5/22). If we offer a time series replacement, which one? How do we support the additional persistence needs in metadata, app services, etc.? Do we have multiple databases? Does our database abstraction make it possible to put in a time series database?
- External control (start/stop/restart) of services (per Levski planning 5/22) - now that System Management Agent and executor are deprecated, what will our solution for these controls be in the future? Use Portainer, Docker CLI, snap CLI, etc.. The GUI and other 3rd party tools would like to see an alternative.
- MQTT 5 (per Levski planning 5/22) - Add support for MQTT features (in particular its bridging feature, topic aliases, and better response or "return codes" for various errors.
- Global configuration - there is a lot of config duplication across services which requires a lot of config to be touched when only one item needs changed (examples: topic name, log level). Issue is where would the common configuration go and how would it be made available? Would it work with or without Consul and with/without security. Would it work the same in containerized, snap and non-containerized environments. Per Kamakura planning (Nov 21), this issue is backlogged for now, but should be addressed before the next major release (v3).
- There is a general concern that EdgeX is too big for some edge environments. This is largely due to the fact that enterprise/cloud products are used for security and configuration/registry. Are there alternative products? Could the open source groups supplying Consul, Kong, Vault be partners in developing edge-sized versions of their products? It is estimated that we only use about 10% of the functionality. Per the Kamakura planning meeting (Nov 21), this issue was backlogged for a future release. **Good first project** for those new to EdgeX: research alternative products or ways to reduce the size of the existing large services. Alternatives to Kong suggested are Tyk, Consul API Gateway, NginX.
- SMA and the system manage executor are deprecated. The features of the SMA only are supported in some environments. Most deployment orchestration tools/platforms or even the OS can provide memory and CPU utilization for the services. Start/stop/restart can also be done by other things when running containerized, for example. Configuration information is available through Consul when it is running. So what replaces SMA going forward? Does anything replace it? What features need to be provided by EdgeX directly versus by the adopters choice of management system? If/when considering an SMA replacement, these additional needs/enhancements should be explored:
    - System management - storing system metrics locally
    - System management - setting configuration (finding ways to differentiate read-only versus read-write props)
    - System management - actuation based on metric change (a "rules engine" for control plane data)
    - System management - add alerts and notifications (service down, metric above threshold, etc.)
    - System Management Agent to store configuration as a config provider in a lightweight configuration store (as a replacement of Consul).
- There are some requests from adopters to expand support notifications. Namely, to send notifications via SMS, web sockets or message protocol (like MQTT or Redis Pub/Sub). There is also a request to make using/sending notifications easier. Today, a developer must write code directly into a service to be able to send a notification. Is there a way to do some notifications by configuration? At the very least, better examples need to provided on how to use the notification service. Also needed is a way for the rules engine to trigger notifications. Per the Kamakura planning meeting (Nov 21) this enhancement as been deferred to a future release.
- Full Windows development support - ZeroMQ libraries prohibit building EdgeX Go services on Windows. A work around using a compile directive not to compile ZMQ code has been added. Future work includes looking at use of a ZMQ replacement library or just removing ZMQ all together with the next major release (v3). Per the Kamakura planning meeting (Nov 21), looking for an alternate library and potentially producing a Windows artifact with our CI/CD processes (MSI, Windows exe, etc.) is tabled for a future release.
- Produce a container with multiple service executables (e.g. create a single core container that is core, metadata and command all in one). Discussed at the Kamakura planning meeting (Nov 21) and deemed out of scope for now. There is some consideration that must be given to env vars (like port) and env var overrides and such that make this complex. **Good first project** for those new to EdgeX: conduct some experiments to do this and work out the env var issue. Provide statistics on the possible savings.
    - As a corollary to having multiple services in one container, have one Docker image but multiple containers. This would reduce the image pulls. Also reduce the security/signing need.
- Move the command API and services into core metadata. This would eliminate another service. Per the Kamakura planning meeting (Nov 21), combining these services was discussed and tabled for now. There are reasons that the command service was initially created that may still be needed in the future to include: additional security and access control that might be needed before issuing an actuation command and the need to expand the command service to allow it to send commands to multiple device services and devices simultaneously in the future.
- Improve "things provisioning". Allow for more auto device/sensor discovery and provide information in the profile for how to handle this. Discussed at the Kamakura planning meeting (Nov 21). More research is needed to understand tangible improvements and what "zero touch" means to an adopter.
    - As a corollary to improving provisioning, there is desire to allow EdgeX to receive a package or "chunk" of configuration that includes profiles and device configuration that allows EdgeX to provision (or update) a sensor/device with a single request.
- Integration of EdgeX with Prometheus. Discussed at the Kamakura planning meeting (Nov 21) and deemed out of scope at this time. **Good first project** for those new to EdgeX.
- Add a NATS implementation of the EdgeX message bus abstraction. Requested by at least one adopter and discussed in the Kamakura planning meeting; deemed out of scope for this release.
- Provide for a record and replay device service - as outlined here. **Good first project** for those new to EdgeX.
- Allow device services to filter event/reading before sending to core data or app services.

- Provide app service integration to AI/ML packages like TensorFlow Lite.
- Bring your own Vault - allowing users to potentially run Vault in a cloud (discussed during the Kamakura planning meeting - Nov 21)
- Create a tool or script to create a new device or application service (discussed during the Kamakura planning meeting - Nov 21). **Good first project** for those new to EdgeX.
- ARM 32 support - MongoDB is not available on ARM 32; CI/CD infrastructure is required
    - As MongoDB was removed from EdgeX with the Ireland release, this could be revisited with enough adopter demand.
- Alternate (from Docker/Docker Compose & Snappy) deployment and orchestration options for EdgeX. Today, while the community and users of EdgeX are free to deploy EdgeX as they see fit based on their use case/needs, the EdgeX community provides Docker container images and a docker-compose.yml file to help get and deploy EdgeX. Going forward, it is anticipated that the community will seek and even need alternate means of deploying, managing, and orchestrating the EdgeX micro services. Options include using Kubernetes, Swarm, Mesos, Nomad, to name a few. Additional support may be offered by system management tools or facilities. An edgex-example may be the best way to handle these needs depending on complexity and need.
- Facilitate command information to be supplied and known to the northern edge systems. For example, how would we provide Azure IoT with commands that it or a cloud solution could use to actuate on devices? Azure IoT or other north side system could make a request call to the command service for the information, but this requires it to pull it and not be pushed the information.
- Code signing – how to certify the integrity of the system
- Artifact signing (binary executable, JAR, Docker containers, etc.)
- How to secure EdgeX's devices. How do we make sure a sensor or device is safe to accept data from? How do we onboard/provision a device securely?
- Explore potential use of hyperledger. How can an audit on data collected at the edge by established and tracked (given the limited resources at the edge)?
- Protect data at rest (encrypt the database)
- Protect data in motion (encrypt data passing between services or outside of EdgeX)
- Support privacy concerns (GDRP, HIP-A, etc.)
- Improving EdgeX resiliency in face of issues or non-availability of some resources/services/etc. (typically for core and above services and not device services).
    - Insure appropriate micro services follow 12 factor app methodology (see https://12factor.net/)
    - Allow select services to be load balanced
    - Allow select services to fail over
    - Allow for dynamic workload allocation
    - Allow services to live anywhere and be moved without lots of configuration to be changed in other services
    - Allow services to be distributed across hosts - and across API gateways (requiring service to service communication protection)
- Support truly distributed micro services
    - Allow services to run on multiple host machines
    - Secure distributed EdgeX with reverse proxy
    - Cross EdgeX instance command actuation (ex: device X on EdgeX box A triggers action on device Y on EdgeX box B)
    - Front a collection of duplicate microservices with a load balancer (allow for the microservice copies to scale up or down based on load); allow multiple instances of any microservice (for future load balancing and scaling efforts - today only single instances are allowed)
- Develop a real hardware test environment/playground
- Develop a test environment to test distributed EdgeX
- Support enrichment functions (an EAI concept) in export services (or application services). Allow additional data or information to be added to the flow of sensor data to the northbound side. This might be information about the sensor/device that captured it or information about the commands to actuate back down on a sensor/device.
- Support additional northbound formats (in some cases additional OT protocols)
    - Haystack
    - OPC UA
    - CloudEvents (not provided today because CloudEvent requires too many libraries and would balloon the size of the app service configurable)
- Support additional southside connectors (with appropriate license/open source availability)
    - Profinet/Profibus
    - CANBus
    - LORA
    - IoTivity
    - Zigbee
    - ZWave
- Provide more tooling for the device service SDKs. The original Java DS SDK was command line driven. In the future, generating a new SDK can /should be done from tools such as IntelliJ.
- Allow EdgeX (the entire platform) to be multi-tenant. Data and services can be attributed to individual clients and protected from one-another.
- Downsampling: It is mentioned that the device service may receive from the device new unattended readings (e.g. in a pub/sub type of scenario). In this case, there should be a setting to specify whether we accept all readings or we decide to downsample because the source is pumping data too fast. This is actually a very common scenario when you deal with high frequency sensor packages.
- Data Transformation: This is something we have always considered as a potential in EdgeX – that of a filter, even a small transformation engine between device services and core data. Not a full blown export, but something that serves in a similar fashion and was common across services. We even thought about making it some type of quick CEP/rules engine feeder for those decisions that can't wait to go through the rest of the layers.
- Define security testing and implement the necessary harness to automate security testing.
    - Port scanning (to ensure something hasn't been accidentally left open)
    - Check for weak passwords
    - Test positive and negative access based on access control lists
- Improve binary data support
    - Local edge analytics may be fed binary data and create intelligence from it to allow for actuation at the edge based on the binary data (example: examine an image an detect the presence of a person of interest).
    - Support alternate message formats for service-to-service exchange (Protobuf, XML, etc.)
- Wrap any open source software with an API, or provide common / replaceable library to use by services to communicate with the 3rd party infrastructure (Consul, Vault, Kong, MongoDB, database, etc.). This allows the easier replacement or substitution of the infrastructure in the future.
- Data visualization - how can sensor data be better visualized and analyzed at the edge?
- Automate the generation of the API documents from the code (versus manual creation of the API documentation today)

- Several of the device services were created to prove, conceptually, how to connect to a device using that protocol, but it may not be a full implementation of the protocol API. For example, the SNMP device service implements enough to drive a Patlite and a few other devices, but does not understand all of SNMP.
- Allow services to respond to native init processes (example: as that provided by systemd). The functionality added with the outlined system management APIs should help facilitate systemd calls
- Use of QoS and/or blockchain to prioritize resource usage by certain services
- Automatic code formatting in CI/CD pipe
- Additional language SDKs
    - Device Service SDKs in other than Go or C
    - Application functions SDK in other than Go (C or Python are two requested)
- Tooling for SDKs (CLI, JetBrains or Eclipse plugins, etc.)
- Tooling / UI for Device Profile creation (IOTech is providing its DCT tool to users for free now)
- Secure service-to-service communications

Technical Debt

These are changes to the existing EdgeX microservices or system at-large to address concerns of scalability, flexibility, reliability, maintainability, or otherwise improve the existing system that is otherwise considered inferior design, buggy or otherwise poor quality code. Technical debt may be thought of simply as "areas of the existing code base we need to fix".

- Improve/additional testing
    - Scale testing
    - Soak testing
    - Platform/OS Testing
    - Different configuration testing
- Automate generation of API documentation. The RAML or Swagger/OpenAPI that provides developers with an understanding of the microservice REST calls is maintained by hand. How can the generation and maintenance of the API documentation be more automated and therefore allow the documentation to be more reflective and consistent with the code base?
- Replace poorly supported 3rd party libs (as of Kamakura planning meeting - Nov 21):
    - bitbucket.org/bertimus9/systemstat
    - github.com/armon/circbuf
    - github.com/go-logfmt/logfmt
    - github.com/mitchellh/consulstructure
    - github.com/pascaldekloe/goe
    - github.com/ryanuber/columnize
    - github.com/sean-/seed
- MQTT message bus implementation does not handle different hosts for publishing and subscribing at this time. (Per Kamakura planning meeting - Nov 21)

No Longer Considered

- Declarative Kong - Kong is big and takes up a lot of memory. Declarative Kong would remove the need for Postgres (saving ~157MB image), but the memory consumption of Declarative Kong increases (+20MB) because the data is stored in memory (but save ~45MB memory used by Postgres). It would also make the API gateway read-only (you couldn't create new users/tokens dynamically). Finally, it only supports JWT users. Negatives outweigh the positives and therefore shelved indefinitely.
- Allow for new category of micro services: “Sharing Services” for East/West data exchange with non-Edgex entities (too vague)
- Allow for device hierarchy in metadata model: a device could be a manager for another device while also collecting data itself. Sending a command to a managing device could mean sending a command to all associated devices. (too hard and not enough of a use case/demand to look at. Different for each type of protocol)
- Better support mesh network protocols (Zigbee, BACNet, etc.) where devices know and sometimes manage other devices (see above)
- Command: in order to protect the device from harmful commands, there should be the possibility to set a Min and Max limit for the value that is accepted on every single command. in fact the command service today is rather a hollow simple proxy, but in the future we very much envisioned adding additional security, caching to avoid having to hit the DS when unnecessary, and even grouping command requests for better resource conservation (especially for devices like BLE that get woken up when you hit them). (This is more of a general security DoS type of attack to be explored universally at some point in the future)
- Support for alternate logging formats and/or more structured logging (logging service being deprecated in favor of using standard out and file system - which invites 3rd party tools to be used and any format supported by those tools)
    - XML, CSV vs JSON
- Integrate to edge software/agents (not enough demand or use case)
    - AWS Greengrass
    - Microsoft IoT Edge
- 0MQ export function (not enough demand or use case)
- Support additional northbound endpoints and protocol types. (provide examples but not implement in services; like we do today for Azure, Amazon, etc.) Examples include:
    - Tencent
    - Alibaba
    - IBM Watson
    - IoTivity
    - SAP HANA
    - DDS
    - AMQP
- Educational Assistance (too vague)
    - Meetup support
    - Hackathon kits/support
    - Deeper documentation - especially around on-boarding

    - Provide more example code (Device Services, Application Services, etc.
- SDK alignment – can / should the DS and Application Functions SDKs be more aligned (design, usage, etc.)? (To the extent possible, this is being looked at for filtering and some other functions. Otherwise too vague and with no clear objectives)
- Implementation of message bus alternative for intercommunication between microservices as an alternative to REST. (Being accomplished where there is a need. Also have go-mod-messaging which makes it possible to institute where needed.)