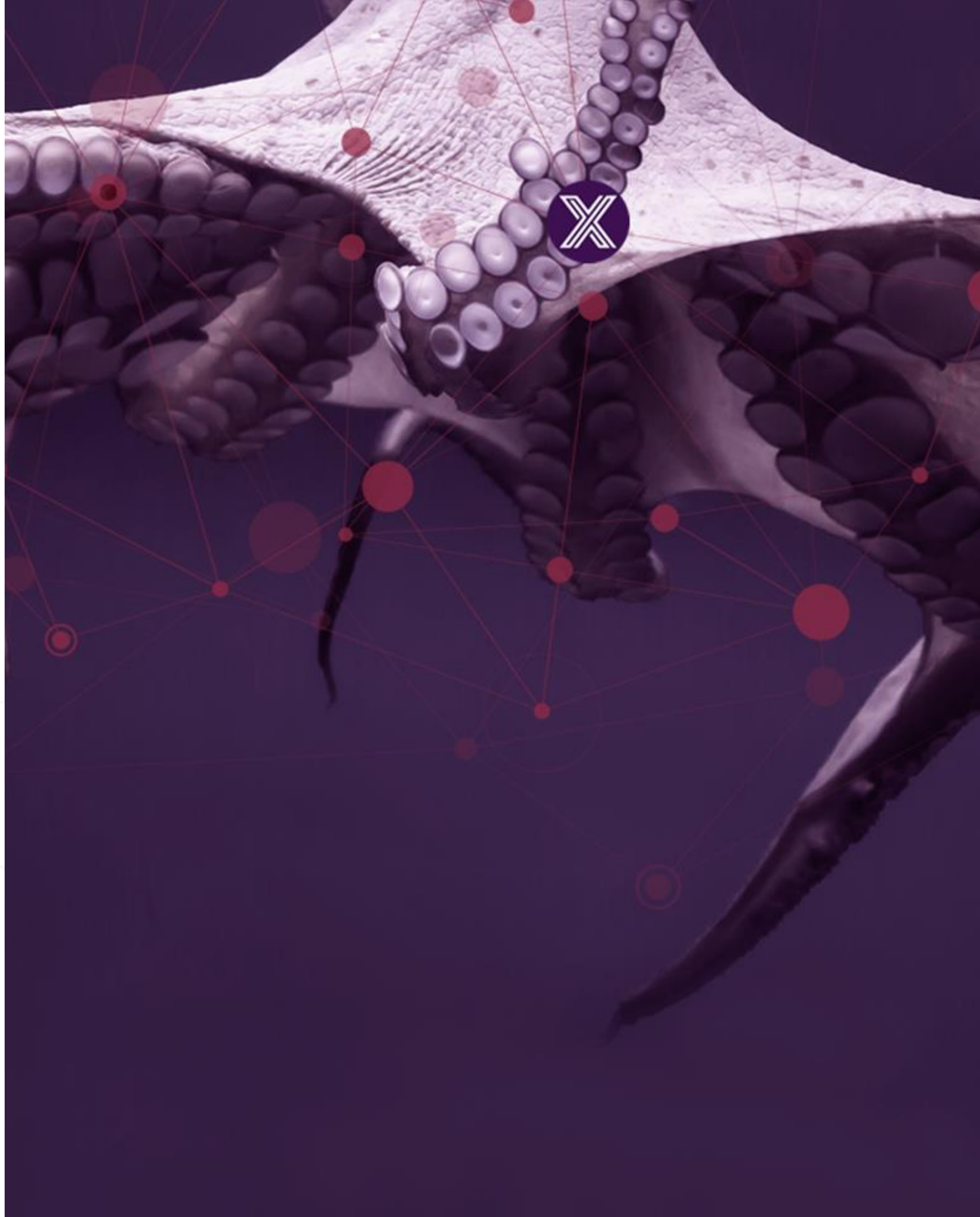
EDGEX FOUNDRY™

Commerce Project Meeting

June 4th, 2019



Rough Agenda

1. Roll Call and Opens
2. Review previous Opens
3. Data flow conversation

Roll Call

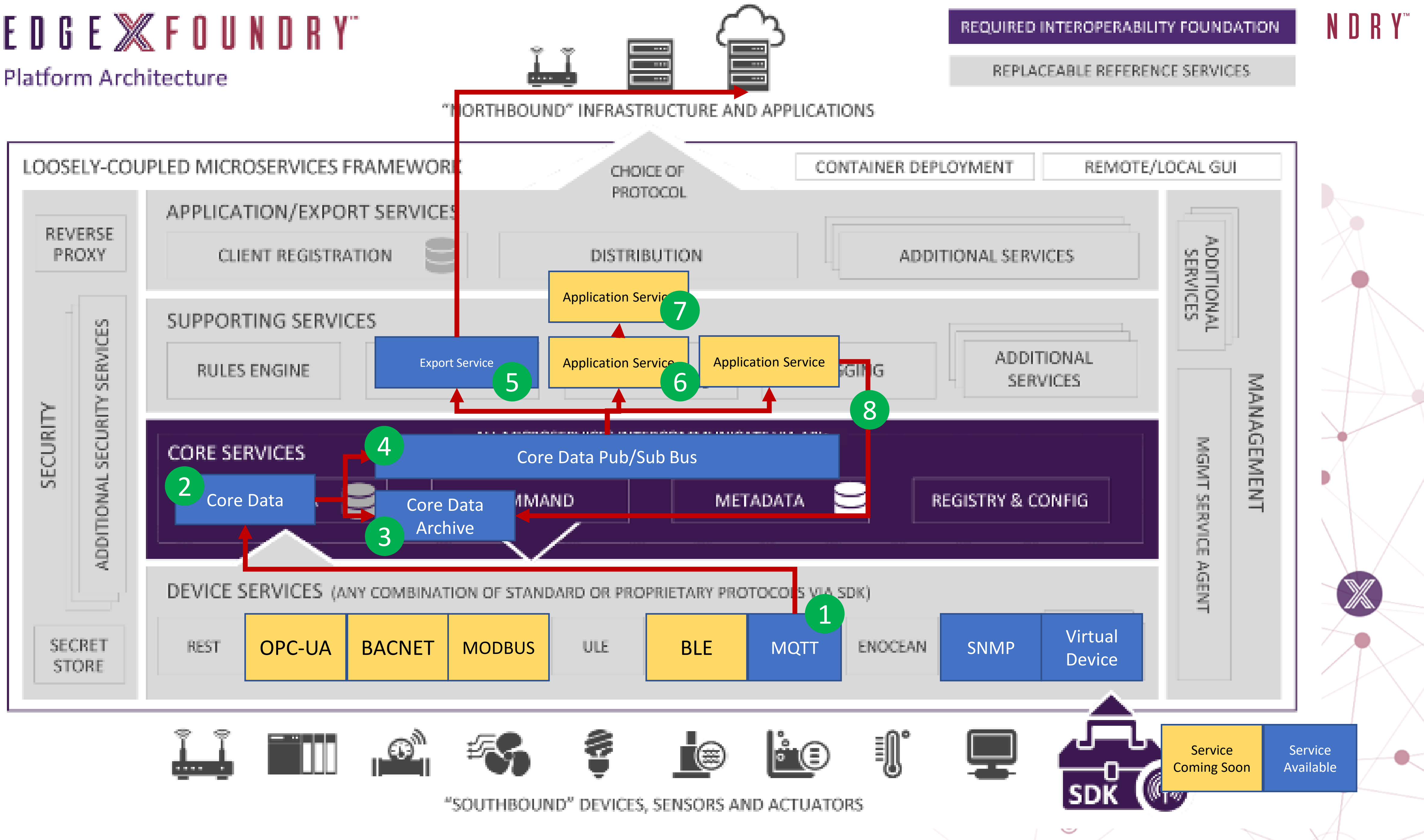
- Brad C (Intel)
- Doug Migliori (ControlBeam)
- Lenny Goodell (Intel)
- Henry Lau (HP)
- Toby Mosby (Intel)
- Torrey Frank (Intel)
- Camilo Dennis (Intel)
- Jim White (Dell)
- Jason Shepherd (Dell)
- Paul Zyskowski (Intel)
- Brad Kemp (Beechwoods)
- Samir
- Eno
- Sam Kaira (Intel Industrial)

Previous Opens

Data Flow

- Purpose: Let’s walk through the path that data from a device takes, to a cloud, and back, and lets throw in a command, too

EDGE X FOUNDRY
Platform Architecture



Data flow walk through from previous slide

1) One of the out of the box Device Services is the MQTT Device Service –

  1) Through configuration can point at a broker, subscribe to a topic on an open broker. There are multiple topics in reality, for push/pull and command/response.
  2) On receiving data, the device service invokes an API on Core Data – "emitting" the EdgeX event. Event contains an envelope (including timestamps), and reading(s) (the payload). In the MQTT DS, any data read off the topic is a single "Reading" in the event. Readings are key/value pairs (name is value descriptor, and value e.g. "temp" and "72"). Events may have multiple readings. The value descriptor can be decoded later when the event has to be utilized. Units are encoded in the value descriptor name and stored in the Core Data metadata. Value descriptor blows out into metadata about value, including datatype, units, min max
  3) With MQTT DS, DS can place new value descriptors into core data for data it expects to be shipping (via configuration TOML (set up auto events, etc) and device profile YAML files (defining resources, value descriptors))
  4) *Area to investigate: where in the stack do blobs of unknown data get inspected, mapped to known types.*

2) RESTful call into Core Data API.

3) The event is typically recorded to the database*. Non-binary event types are recorded to the database, however binary Reading values are not recorded. For example, if a reading type is application/cbor, the value will not be recorded. Event reading has a binary field and non binary field – binary fields are not stored

  1) *Core Data does have a general configuration flag to not archive –anything- across entire system, which increases performance for streaming applications

4) Core Data now publishes to a topic on a semi-private pub/sub broker. There is an option to publish solely to an endpoint such as to a rules engine.
  1) Core Data can be configured to published to multiple topics, but every event is published to every topic. The default is the "Events" topic per the abstract message bus.
  2) Note: there is no security or permissioned data sharing at this time.

5) Export services are available today and offer configurable export capabilities via MQTT, REST, and similar. Every call is invoked once per event, there is no batching. An Export Service subscribes to the Core Data bus and receives every event published on the bus.

6) Application Services are new to Edinburgh, and are peers of Export Services, and also subscribe to the Core Data bus/topics.
  1) In Edinburgh we have a duality with Export Services and Application Services existing, but with some out of the box Application services can retire Export Services for Fuji release.
  2) Application Services are built upon Application Functions SDK, which provides built in functions to build multi-stage pipelines (within an application service) to filter, process and/or export data. Application Services must be coded as there are no off-the-shelf services. The SI or ISV authors an application service which subscribes to the Core Data event topic. With each event received the SDK functions allow filtering of data. The AS receives all events, so should put filter functions upfront to avoid unnecessary processing further down the stream.
  3) Application services are the natural location to perform higher level edge analytics, like rolling windows, persistence, etc. Perhaps the system is set to not log data in Core Data database because data is primarily video frame data, so then an application service could choose to log data in its own DB of choice.
  4) App and Export services utilize Core Data and Metadata to understand value descriptors and readings. Various ways to push metadata.
  5) We can author/deploy multiple app services in a single deployment, but all will receive all data if subscribed to core data.

7) Application services can be chained, so that one feeds its output events directly to the input events of the next application service.
8) App service or the export service is responsible for marking data as consumed/exported, so that a scheduler can clean up the garbage.

Opens from conversation

- Opens: how to handle cloud data
  - Cloud data (eg enterprised data) - could be a device service (think internet as a temperature sensor). Other enterprise data might be coming as a command service, or something else.
  - Cloud as a sensor
  - Cloud as a reference
  - Cloud as a streaming data source
  - Retail: SKU to UPC mapping (this is a fairly static reference set)
- Samir: device array of individual device – consuming raw device data
- App service – fusing multiple devices, creating higher level events as a device service.
- App Service that fuses raw data into insights – hack: write into MQTT device service topic, or use Core Data APIs directly, or write PUT a command to a device service.
- Investigate: recommended ways we push data back into Core Data from an Application Service?
- Future: do we filter device data before core data
- Core Data can validate data based on metadata (device and value descriptors match). OR turn off