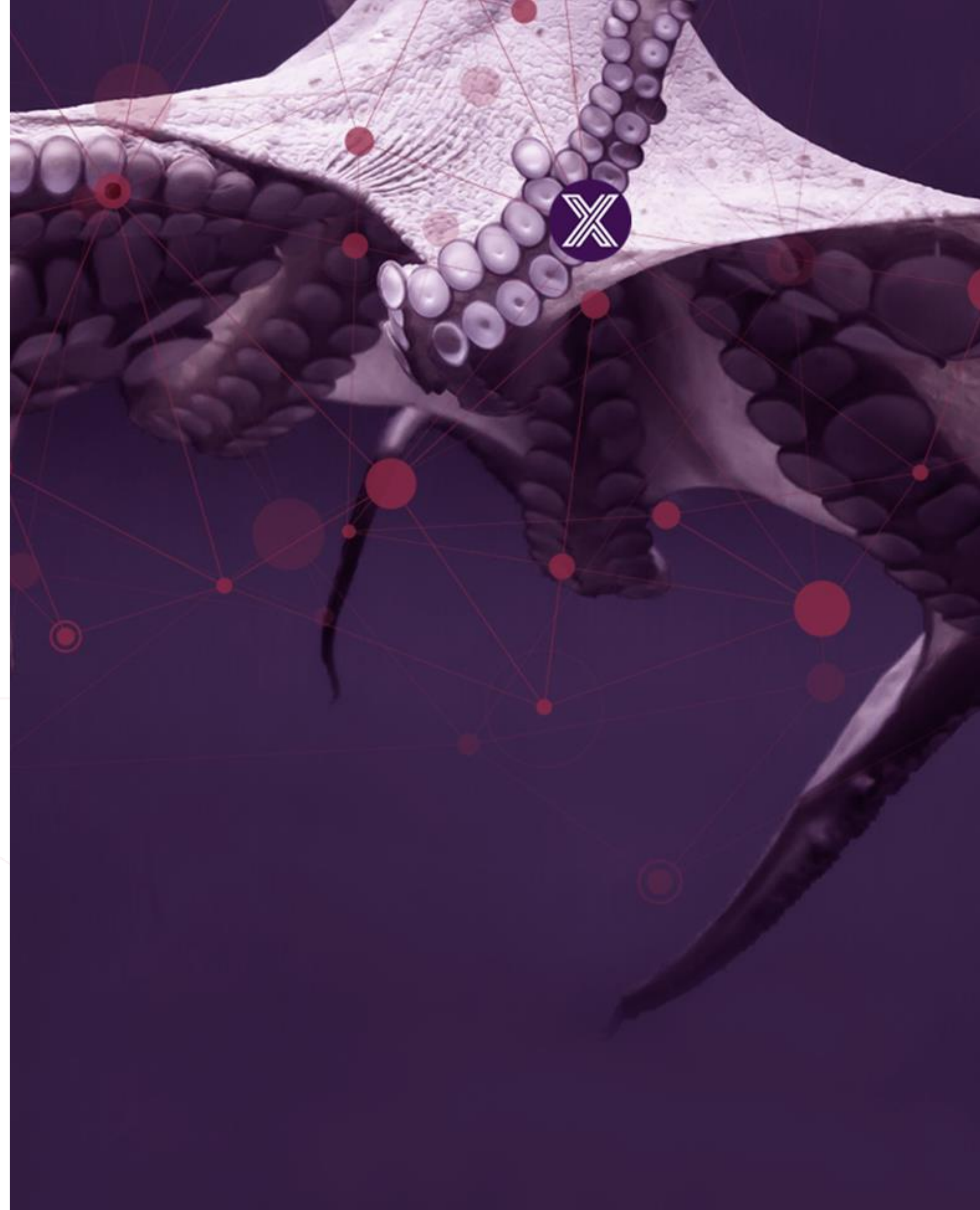
EDGE X FOUNDRY™

Intro and data mgmt. suggestions

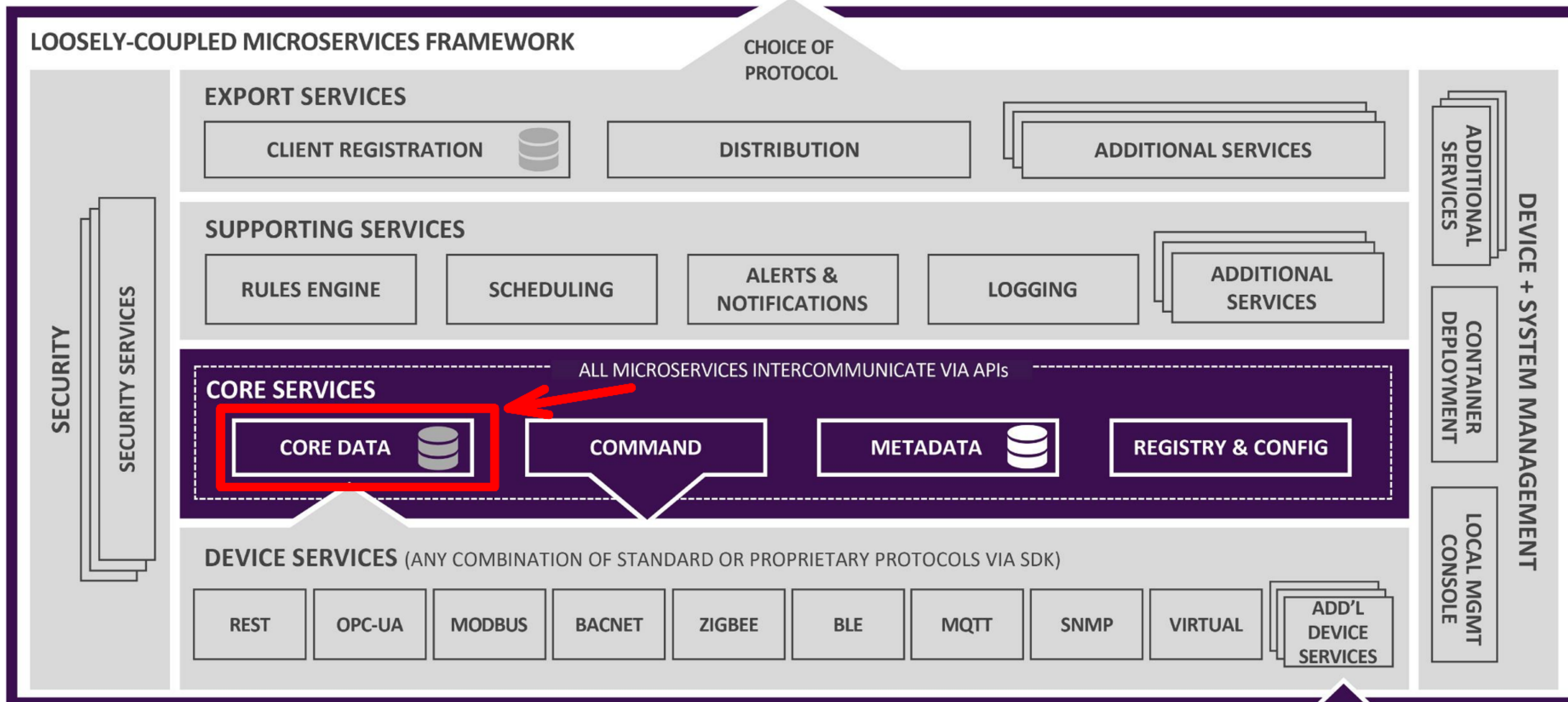


Demo Time!!

May the demo gods be with us

EDGEXFOUNDRY

Big Picture and Core Data



Core Data’s Role

- Provides a centralized persistence facility for data readings collected by devices and sensors.
- Device services for devices and sensors that collect data, call on Core Data to store the device and sensor data on the edge system (such as a gateway)
  - Allows for store and forward technology
  - Supports actuation decisions at the edge
- Data is stored until…: opportunity for data managment
  - It can be moved "north" and exported to Enterprise and cloud systems
  - It is “scrubbed” to make way for new sensor data
- Provides an API that other services can use to access the historical data : data management should be able to access the data in a better way
  - Should be used sparingly as not to impact data collection
- Could provide a degree of security and protection of the data collected by devices and sensors while the data is at the edge
  - Could allow data to be encrypted at rest or in motion

Core Data Makeup

- Created with Java/Spring Framework/Spring Boot/Spring MongoDB
  - Uses Spring MVC for REST communications
- Message pipe connects Core Data to Export Services and/or Rules Engine
  - Uses ZeroMQ by default
  - Allow use of MQTT as alternate if broker is provided
- An alternate implementation of Core Data using SQLite in place of MongoDB was created at Dell
  - Took less than a week to implement and required no additional micro service changes
- Dell has created a partial Go Core Data replacement
  - Working to open source that code later this summer
  - Again with no affect to the other service APIs

Core Data Sequence

Device | Device Service | Core Data | Meta Data | Database | Message Bus

Data pulled (periodically) or pushed (asynchronously) from device/sensor *[device protocol dependent]*

Normalize to EdgeX Event/Reading records

Request to add Event/Reading records associated to the Device by ID *[REST POST]*

Get Device by ID *[REST GET]*

Database not available *[503 service not available]*

device or null

Device ID not known *[404 not found status]*

Insert new events/readings

Database not available *[503 service not available]*

Event/reading identifiers

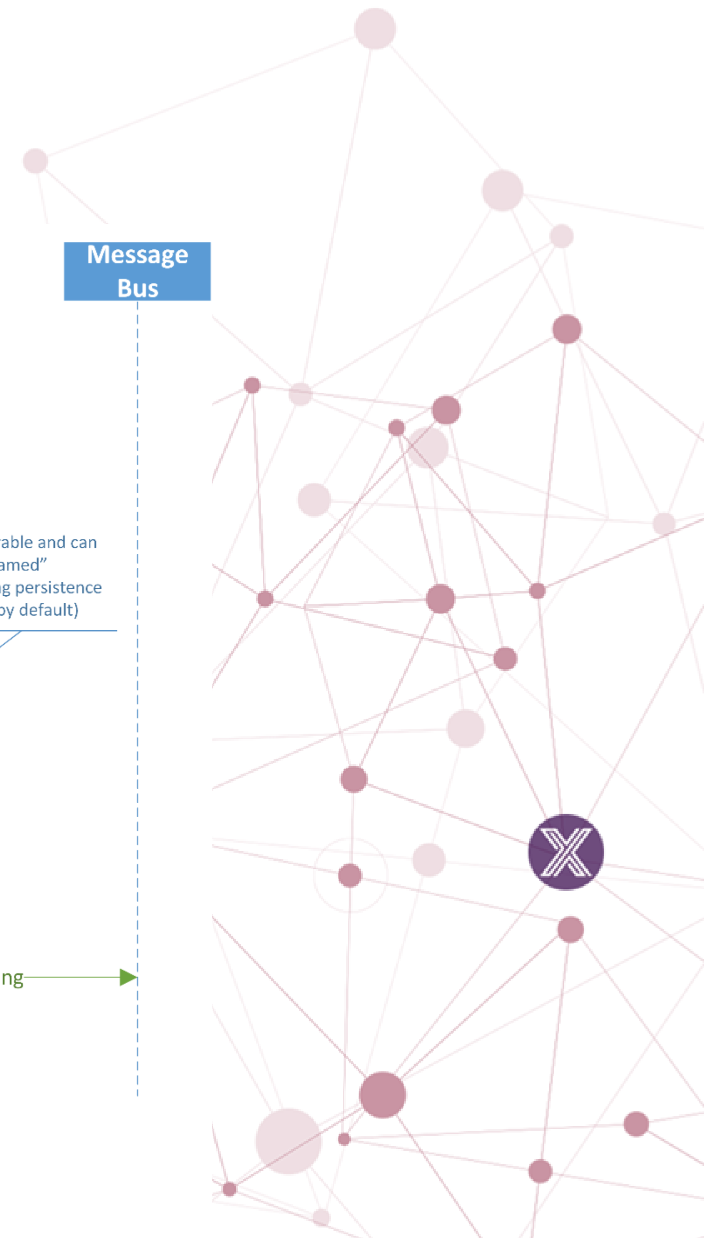
Send Event/Reading to message bus for additional processing

Event ID *[HTTP 202 status code & database key]*

In exception cases, the device service should retry

Configurable and can be "streamed" bypassing persistence (it is on by default)

Configurable and may not always be on (on by default)



EDGE X FOUNDRY

Core Data Model

<<Abstract>>
BaseObject

id (UUID; database generated)
created (long; timestamp)
modified (long; timestamp)
origin(long; timestamp)

the pushed value indicates when the data has been exported for all client requests (i.e. - when can the event and associated readings be removed)

Event - extends BaseObject

pushed (timestamp)
device (string; Device name)

readings 1 0..*

Reading - extends BaseObject

name (string; ex: CurrentTemp; matches ValueDescriptor name)
value (untyped; ex: 55)
pushed (timestamp)

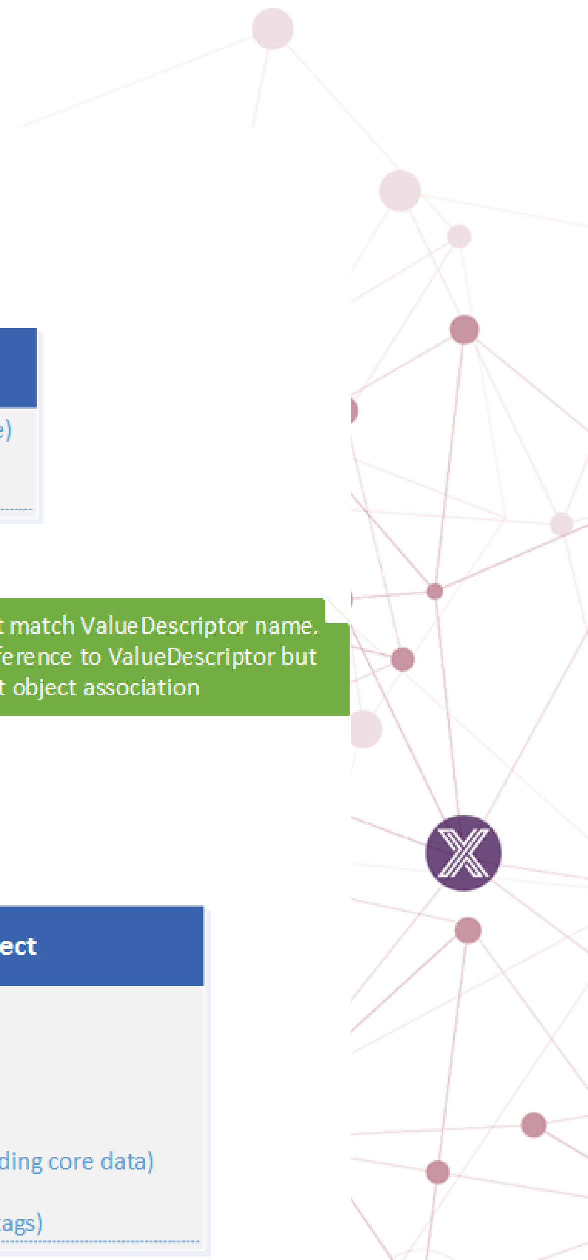
device must match Device name but is not a direct object relation - just a name reference

name must match ValueDescriptor name. name is reference to ValueDescriptor but not a direct object association

0..* name 1

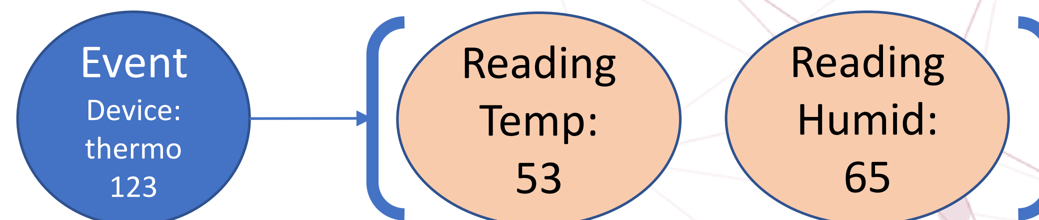
ValueDescriptor - extends DescribedObject

name (string; unique; ex: CurrentTemp)
min (untyped; minum value allowed)
max (untyped; maximum value allowed)
type (Type enum)
uomLabel(string)
defaultValue(untyped; used for UIs; not for seeding core data)
formatting(string; following printf convention)
labels (array of String; ex: [BACNet, temp]; old tags)



Events & Readings

- Events are collections of Readings
  - Associated to a device
- Readings represent a sensing on the part of a device/sensor
  - Simple Key/Value pair
  - Key is a value descriptor (next slide)
  - Value is the sensed value
  - Ex: Temperature: 72
- Event would need to have one Reading to make sense
- Reading has to have an "owning" event



Value Descriptor

- Provides context and unit of measure to a reading
- Has a unique name
- Specifies unit of measure for associated Reading value
- Dictates special rules around the associated Reading value
  - Min value
  - Max value
  - Default value
- Specifies the display formatting for a Reading
- Reading key == Value Descriptor name
- In MetaData, Devices use Value Descriptors to describe data they will send and actuation command parameters/results

```
name: temperature
description: ambient temperature in Celsius
min: -25
max: 125
type: I  (I = integer)
uomLabel: “C”
defaultValue: 25
formatting:"%s“
labels: [“room",“temp"]}
```

Core Data REST APIs (categorized)

- Event APIs
  - POST – new Event with associated Readings
    - Also a PUT to update, but rarely if ever used
  - DELETE – should really only be used by data clean up facilities
  - GET's galore to query for Events, by
    - id
    - Associated Device
    - timestamp (range start & end)
    - Associated Device and Reading with particular Value Descriptor
  - GET Event count (good debug/checking mechanism)

Core Data REST APIs (categorized)

- Reading APIs
  - POST, PUT, DELETEs should only be used by internal facilities
    - But not currently blocked
  - GET's galore to query for Readings, by
    - id
    - Associated Device (via Event)
    - uomLabel (of associated Value Descriptor)
    - label (of associated Value Descriptor)
    - type (of associated Value Descriptor)
    - Timestamp (via Event)
  - GET Reading count (good debug/checking mechanism)

Core Data REST APIs (categorized)

- Value Descriptor APIs
  - POST, PUT, DELETEs
    - Checks to make sure they are not associate to existing Reading
  - GET's galore to query for Value Descriptors, by
    - Id
    - Name
    - Associated Device (via MetaData Query)
    - uomLabel
    - label
    - type
- Ping (good debug/checking mechanism)

EDGE X FOUNDRY™

Import / Unique Core Data Config Options

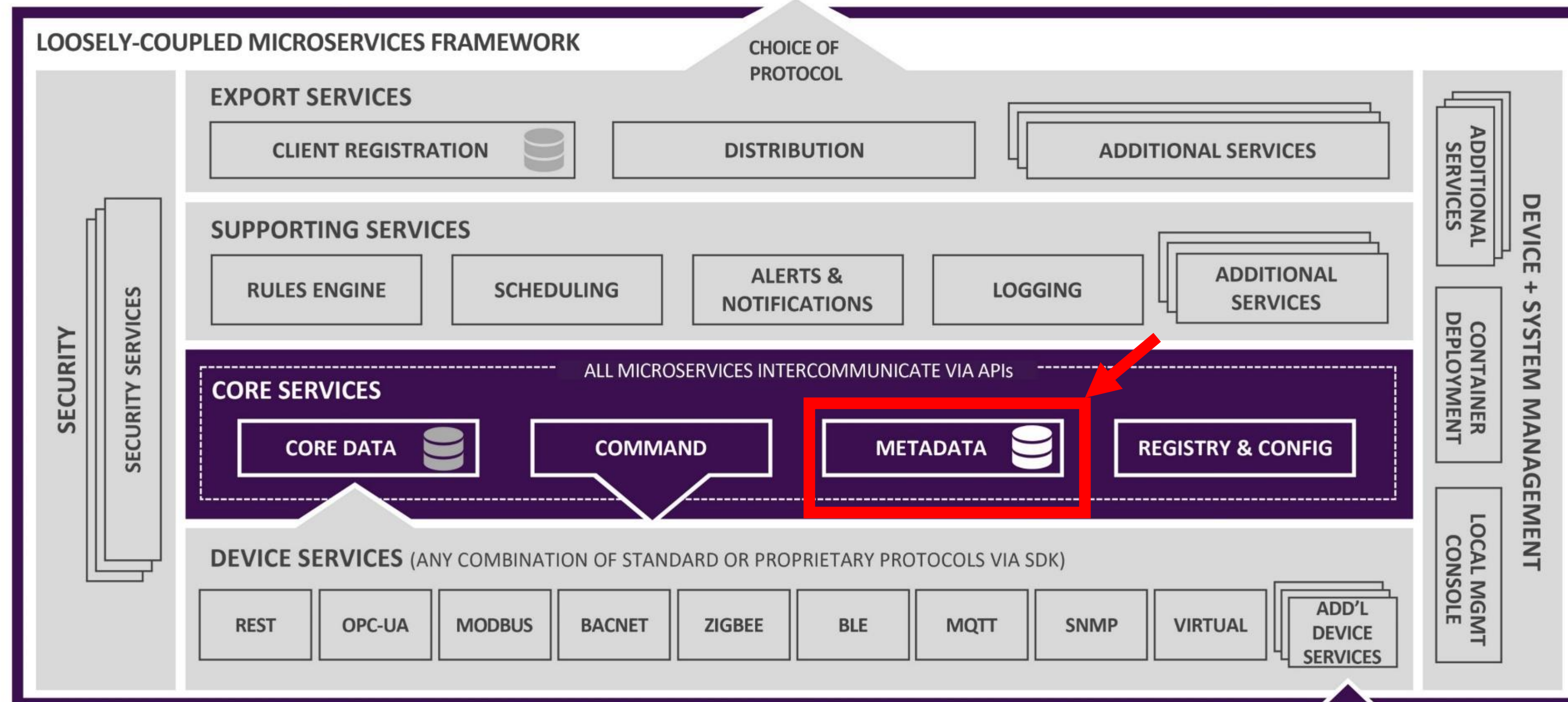
- Per application.properties (or via Consul Config/Registry service)
  - metadata.check (false) – allows you to turn on check of Device existence in MetaData with each Event/Reading POST
  - addto.event.queue (true) – turn off post of new Event/Readings to export services & rules engine
  - persist.data (true) – turn off the storage of data in the database (Mongo) turning Core Data into a streaming service
  - msgpub.type (zero) – send Event/Readings to export services via 0MQ (alternative is via MQTT)
  - Database configuration (location, user, pass, …)
  - Message pipe configuration (location, user, pass, …)

- Other standard config options
  - Service port
  - Location of associated micro services
  - Log levels
  - Read limits

https://github.com/edgexfoundry/core-data/blob/master/src/main/resources/application.properties

EDGE X FOUNDRY

Big Picture and Meta Data



Meta Data's Role

- Is the repository of knowledge about the devices and sensors and how to communicate with them that is used by the other services, such as Core Data, Command, analytics, etc.
- Specifically, Metadata has the following abilities:
  - Manages information about the devices and sensors connected to, and operated by, EdgeX Foundry
  - Knows the type, and organization of data reported by the devices and sensors
  - Knows how to command the devices and sensors
- Meta data does not…
  - Do any data collection, but it knows what data is collected by which devices and which device services manage those devices
  - Issue commands, but it knows the commands that can be issued to any device
- When Meta Data first comes up, it knows nothing and does nothing
  - It depends on Device Services to come up, report their existence and report the devices they have discovered and manage
  - It depends on API calls to identify new device services, devices, device profiles, schedules, etc.
- Could provide a degree of security and protection of the data about the devices and how to communicate with them
- Provides a warehouse of information to ensure incoming sensor data conforms to expectations

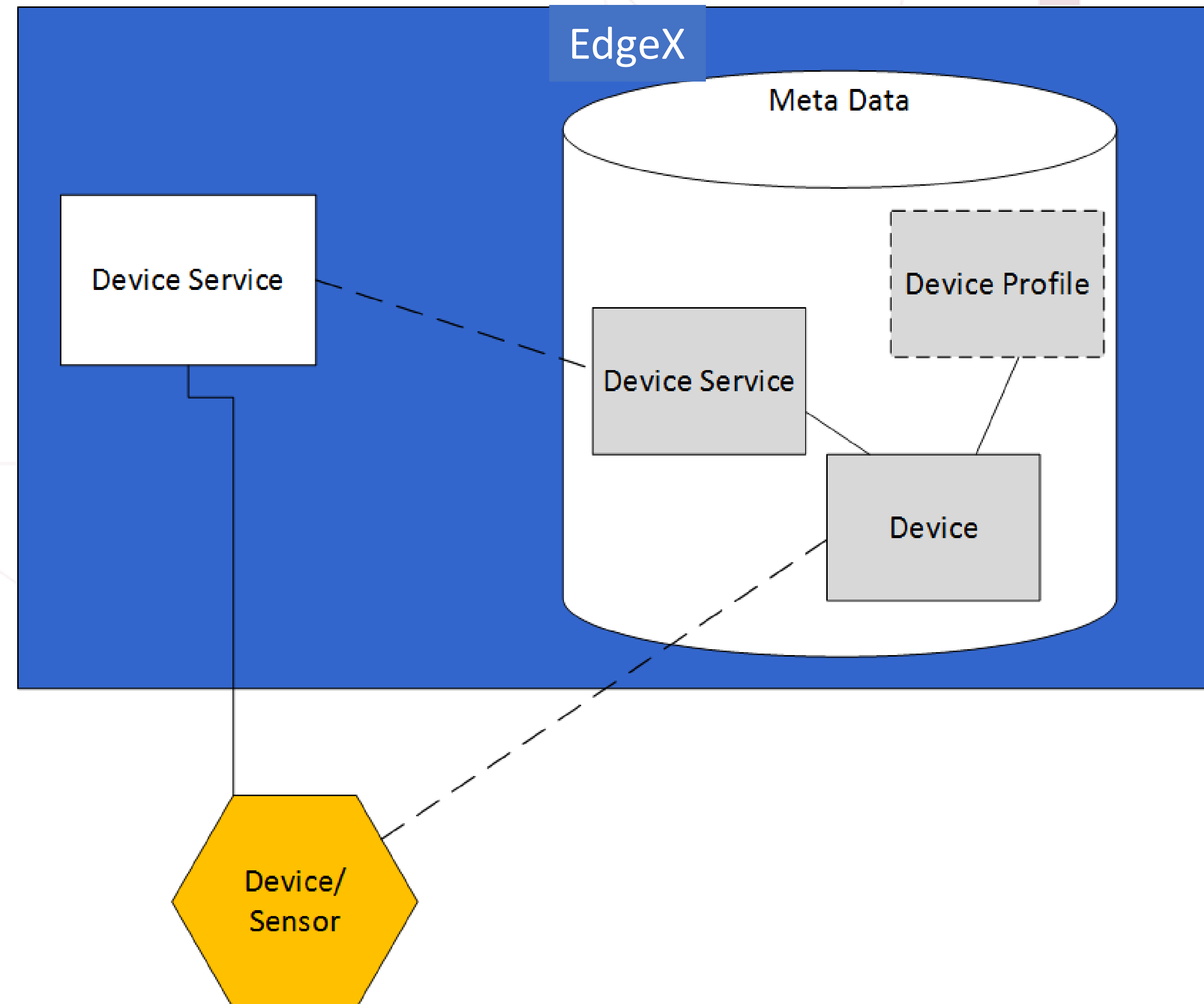
Meta Data Makeup

- Created with Java/Spring Framework/Spring Boot/Spring MongoDB
  - Uses Spring MVC for REST communications
  - MongoDB underneath
  - Core data and Meta data use same instance of MongoDB, but different collections
    - Allows for separation if necessary or use of different DB technology if necessary

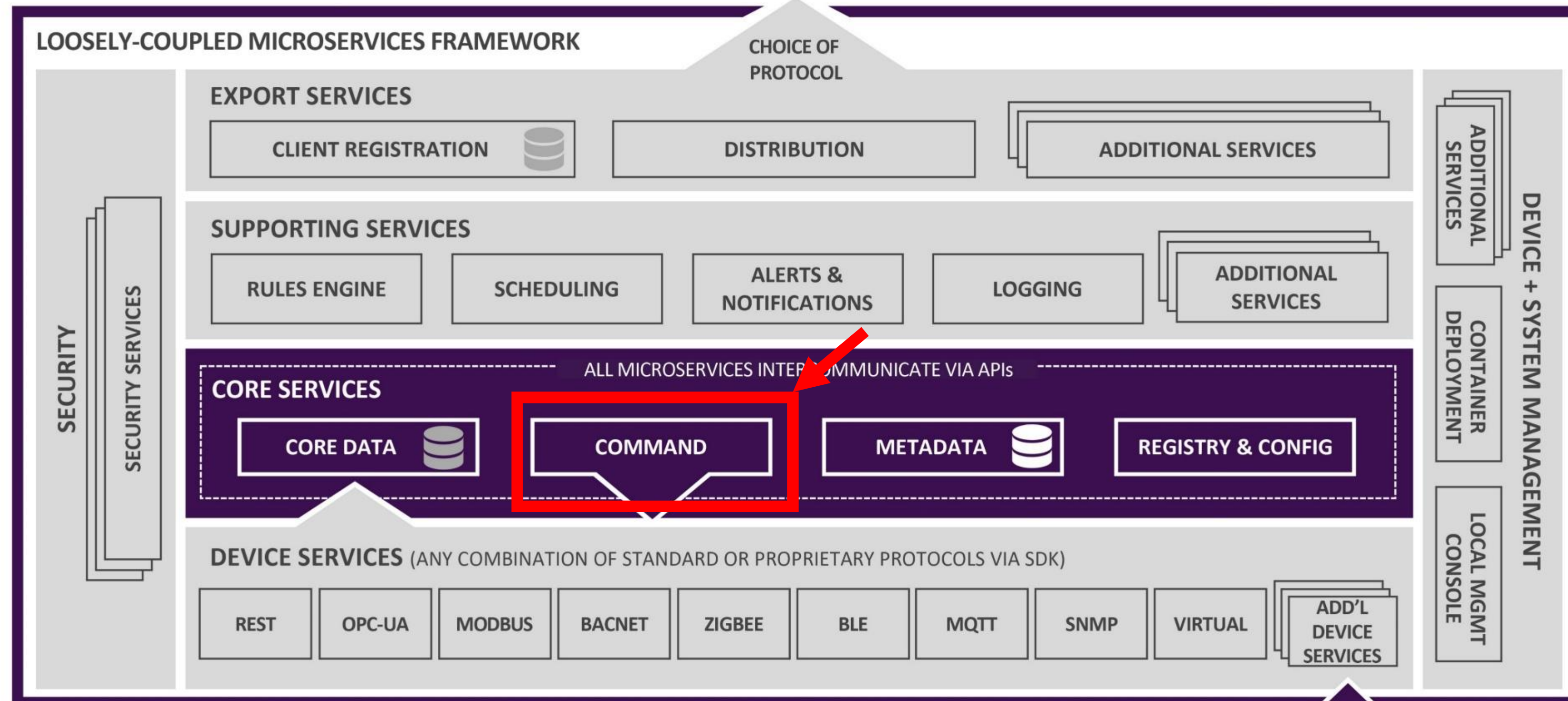
- Dell has created a partial Go Core Data replacement
  - Working to open source that code later this summer or beyond
  - Again with no affect to the other service APIs

Meta Data Triumvirate

- A Device represents a physical device or sensor
  - However, another EdgeX gateway or a system could be a “device”
  - Each device / sensor that is managed by EdgeX Foundry must be registered with Metadata and have a unique ID and name associated to it
- Device services represent other micro services that manage one or more devices
  - Each device is associated to one and only one device service
  - Each device service has a unique ID and name
- A Device Profile can be thought of as a template of a type or classification of Device.
  - A device profile provides general characteristics for the types of data a device sends and what types of commands or actions can be sent to the device
  - A device must be associated to a single device profile
  - More details about Device Profiles and device services next week
  - Each profile has a unique ID and name



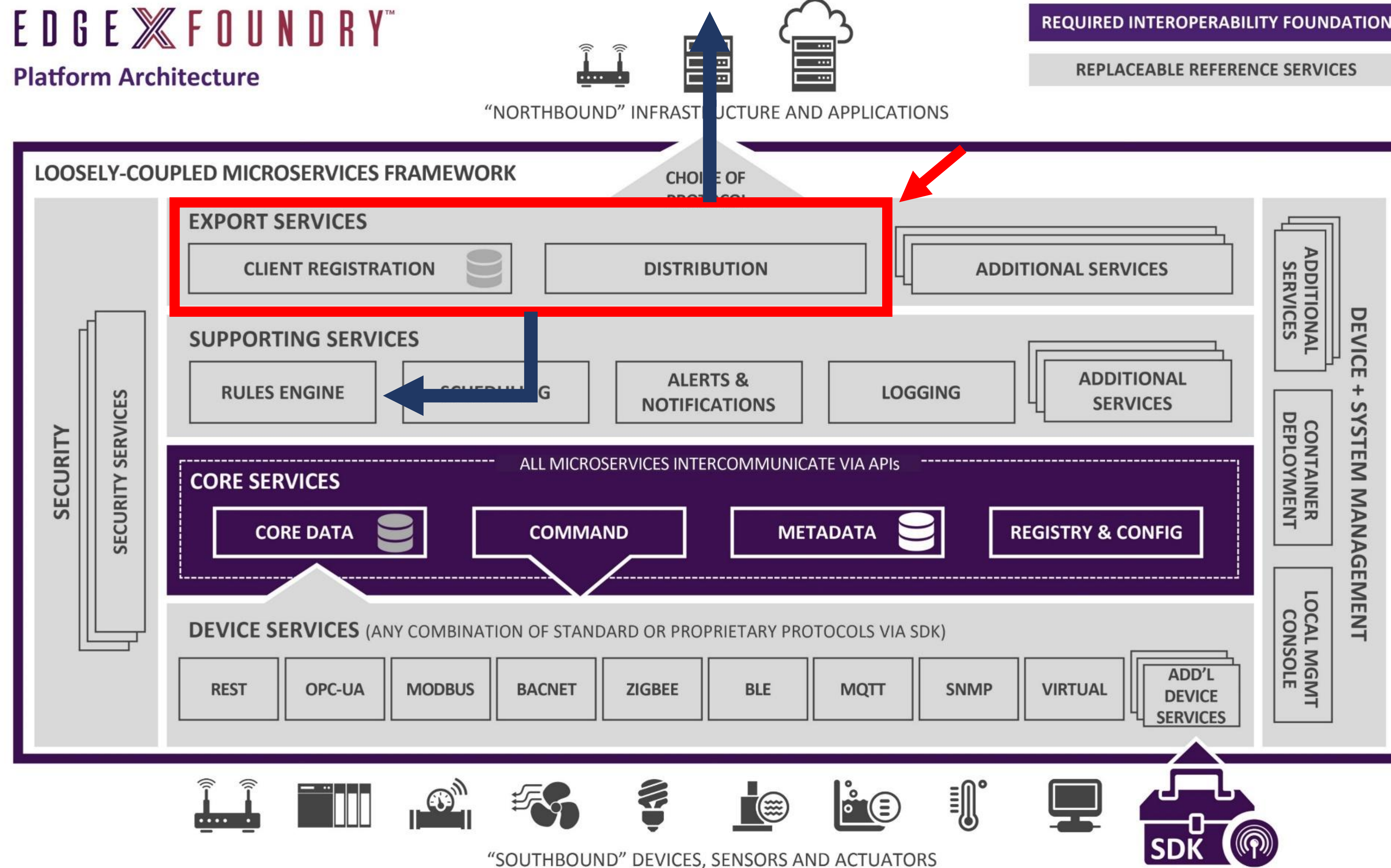
Big Picture and Command



Command's Role

- Also known as the Command and Control micro service
- Enables the issuance of commands or actions to devices and sensors on behalf of:
  - other microservices within EdgeX Foundry (for example, a local edge analytics or rules engine microservice)
  - other applications that may exist on the same system with EdgeX Foundry (for example, a system management agent that needs to shutoff a sensor)
  - To any external system that needs to command those devices (for example, a cloud-based application that had determined the need to modify the settings on a collection of devices)
- Exposes the commands in a common, normalized way to simplify communications with the devices.
  - Commands to devices are made through the command GET, a request for data from the device or sensor,
  - and the command PUT, a request to take action or receive new settings or data from EdgeX Foundry.
- Command does not act alone
  - It gets its knowledge about the devices and sensors from the Metadata service
  - It relays commands and actions to the devices and sensors through the Device Service
  - It never communicates directly to a device or sensor.

Big Picture Export Services

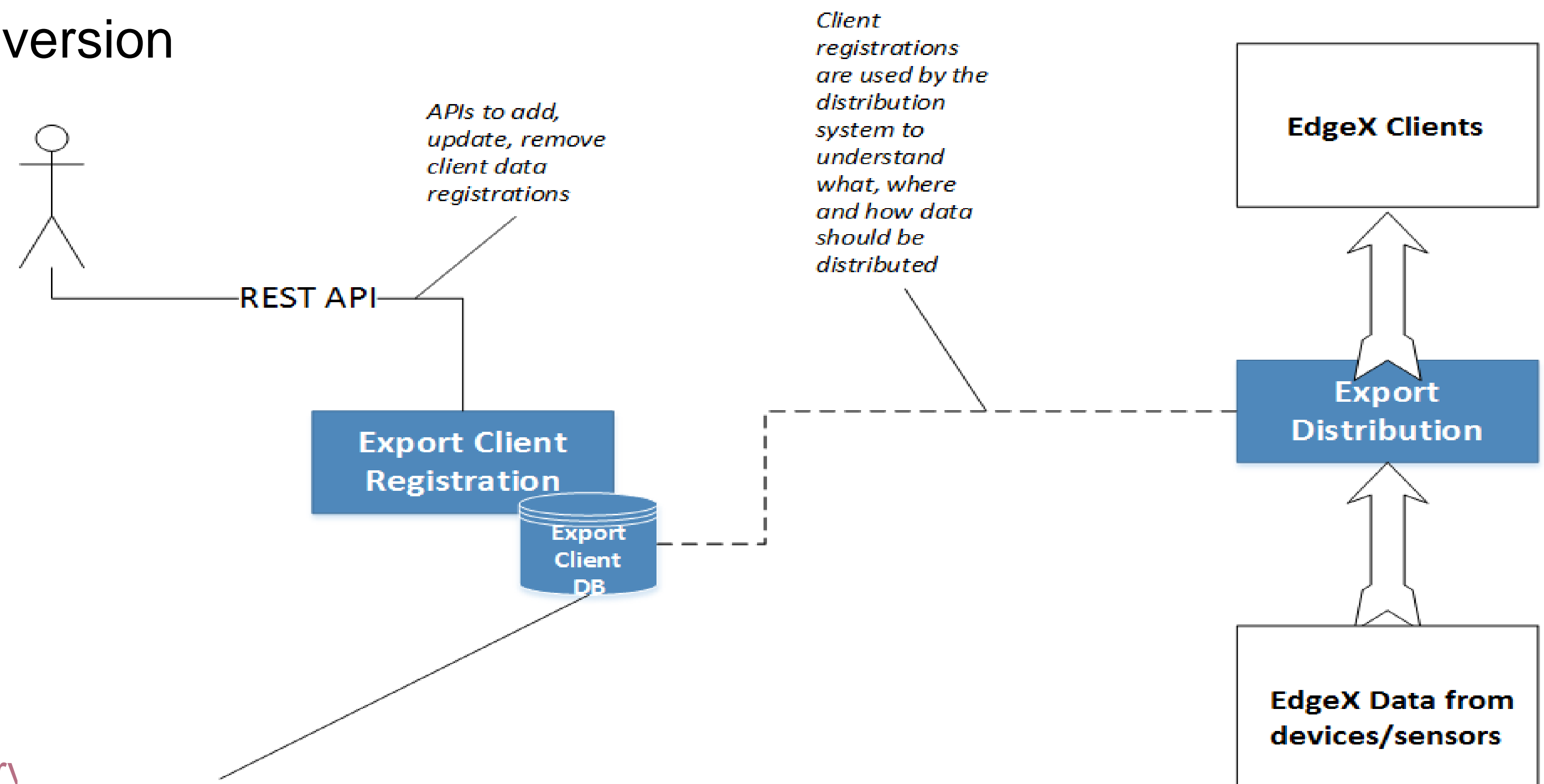


Export Services Role

- Export Services = Export Client microservice + Export Distribution microservice
  - Provides ability to get EdgeX sensor/device data to other external systems or other EdgeX services
  - External systems like Azure IoT Hub, Google IoT Cloud, etc.
  - Other EdgeX services include the Rules Engine microservice or other “analytics” systems/agents in the future
- Export Client – allows for internal or external clients to
  - Register for sensor/device data of interest
  - Specify the way they want it delivered (format, filters, endpoint of delivery, etc.)
- Export Distribution – performs the act of delivering the data to registered clients
  - Receives all the sensor/device data from Core Data
  - Performs the necessary transformations, filters, etc. on the data before sending it to the registered client endpoints

Export Client – Technology and How it works

- Simple Java/Spring MVC application
  - Connections to MongoDB to store client registrations (Spring MongoDB)
  - Completely independent of other microservices (include export distro)
  - Mainflux is building a Go version

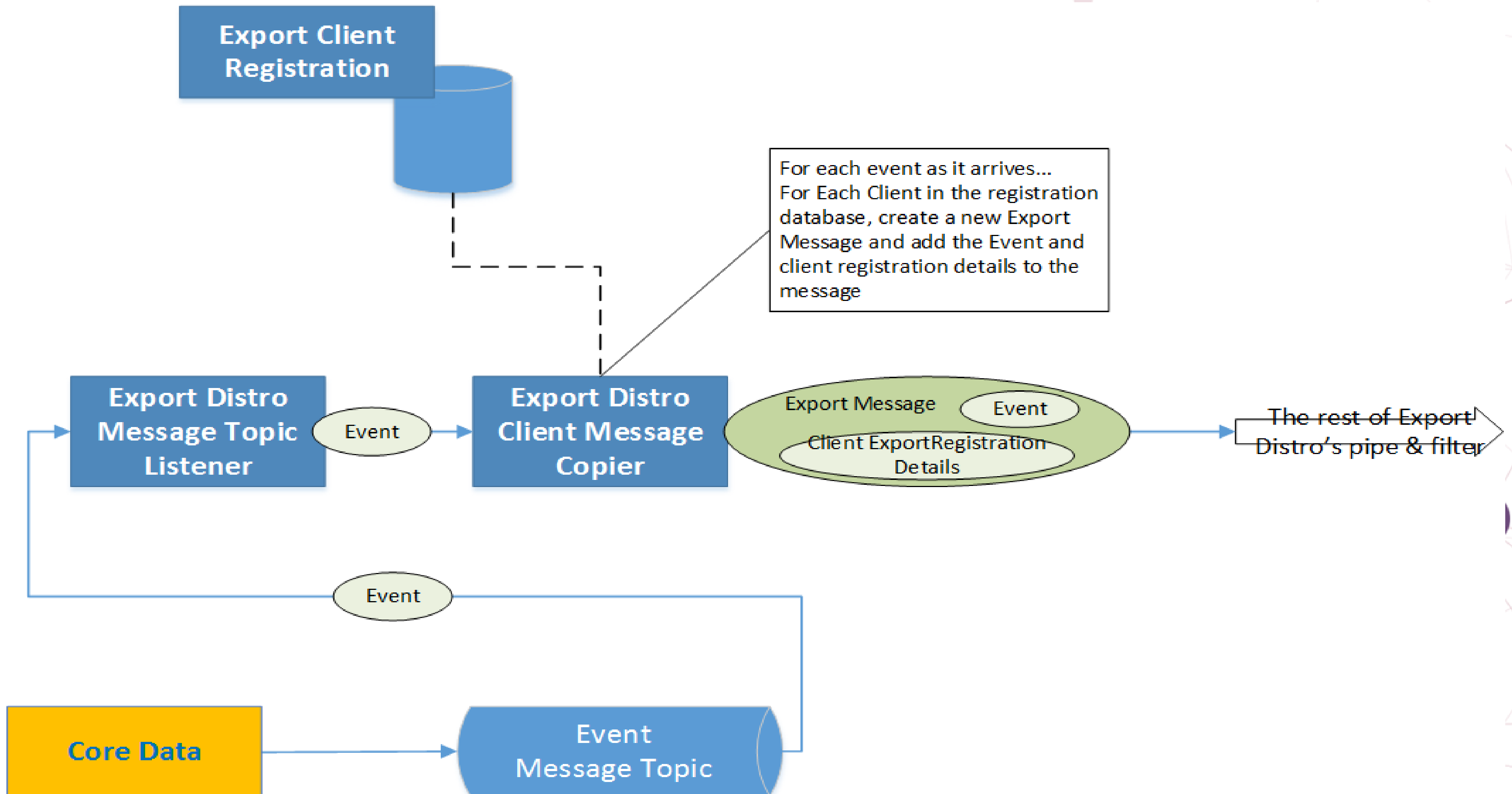


Dell built a simple Angular 2 Web application to provide GUI on top of Export Client

Export Distribution – Technology and How it works

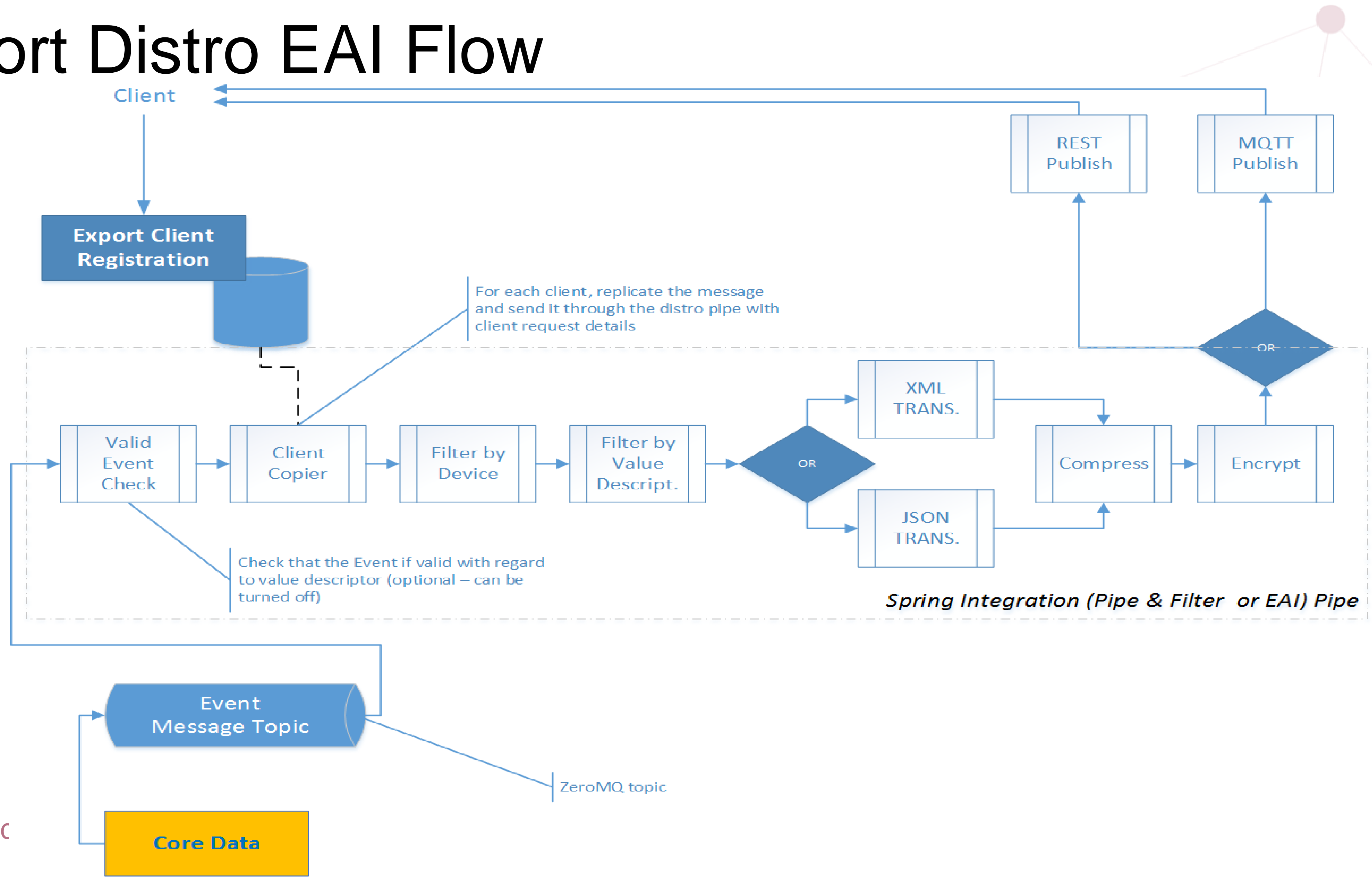
- Java Spring Boot/Spring Integration application
  - Spring Integration is an Enterprise Application Integration framework
  - Follows the EAI patterns (see http://www.enterpriseintegrationpatterns.com/patterns/messaging/)
  - Essentially a Pipe & Filter architecture
- Takes each incoming Core Data Event/Reading (via 0MQ) and…
  - Filters out irrelevant or incorrect data
  - Transforms the data to client's format of choice (XML, JSON, …)
  - Optionally compresses the data
  - Optionally encrypts the data
  - Sends the data to the client's registered endpoint (REST, MQTT, 0MQ, …)

Export Distro Entry Flow



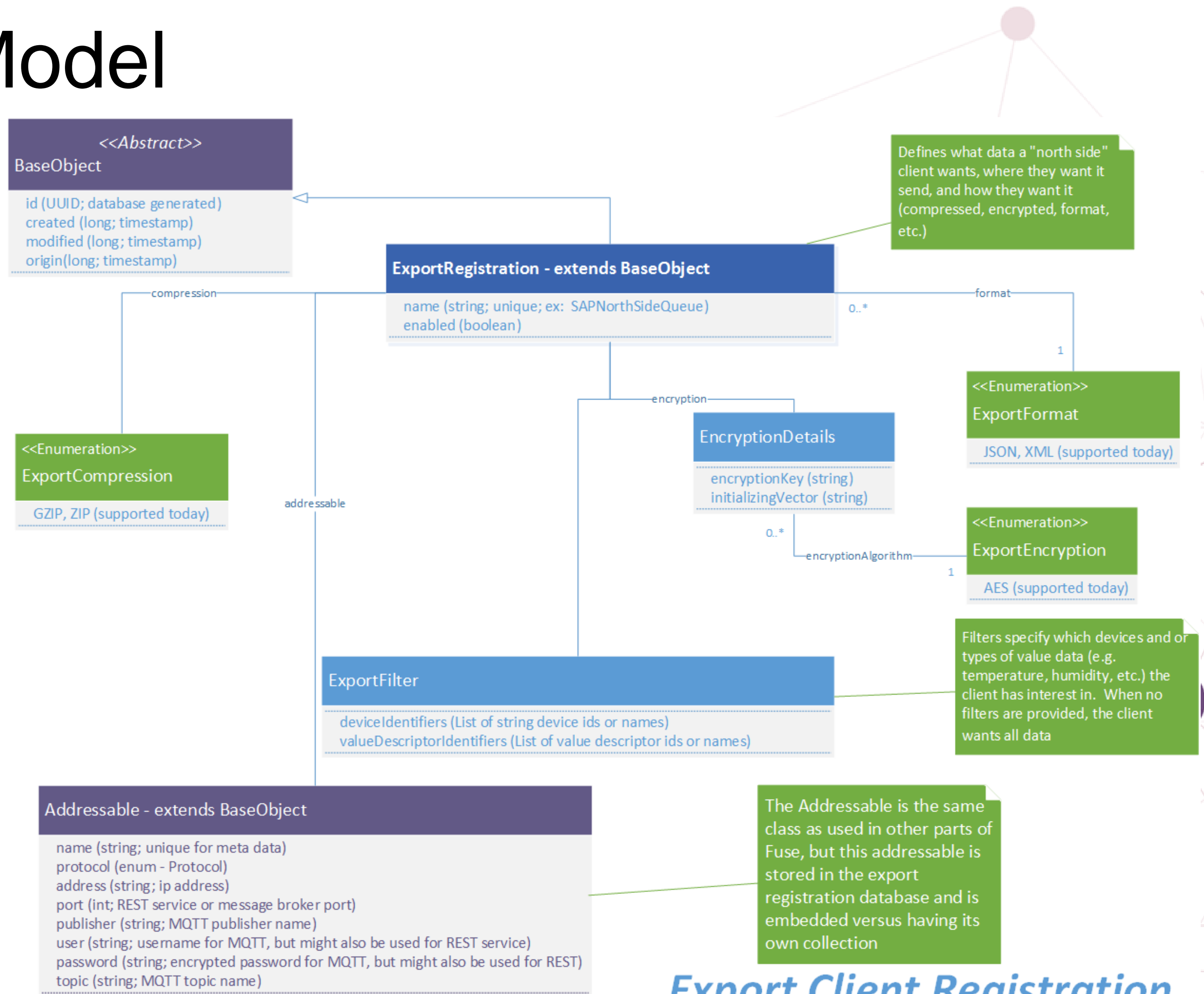
EDGE X FOUNDRY

Export Distro EAI Flow



Export Client Data Model

- Client's register with…
  - Endpoint
    - Held in Addressable
    - MQTT, REST, etc. details
  - Filters
    - In ExportFilter collection
    - By device or value descriptor
  - Format
    - In ExportFormat
    - JSON, XML
  - Encryption
    - In ExportEncryption
    - AES today
  - Compression
    - In ExportCompression
    - GZIP, ZIP



Export Client API

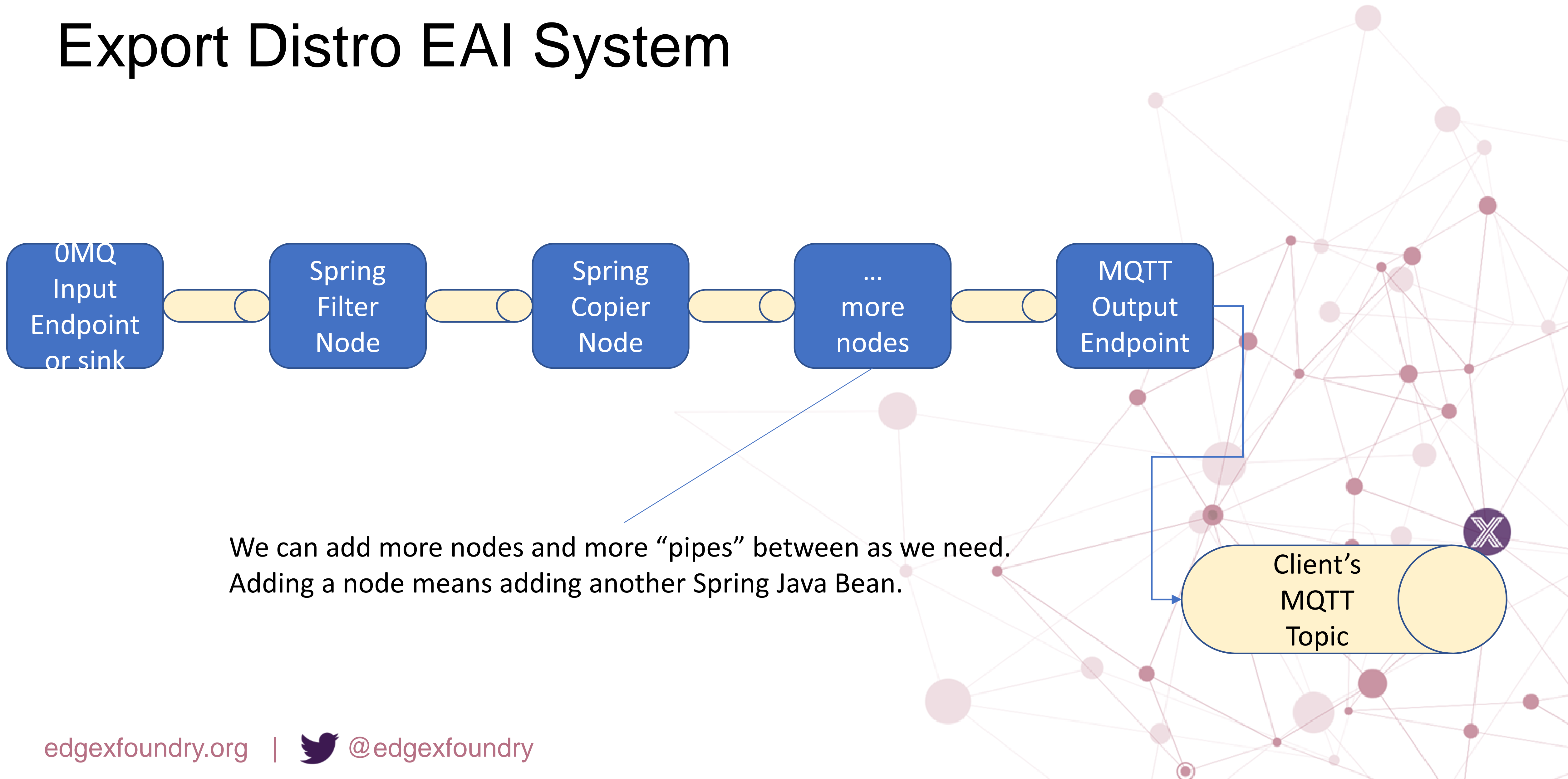
- Export Client API is pretty basic – typical REST resource operations
  - At port 48071 by default
  - Register a new client export request
    - /api/v1/registration (POST) with ExportRegistration object in JSON
  - Update an existing client export request
    - /api/v1/registration (PUT) with ExportRegistration object in JSON
  - Remove (deregister) a client export request
    - /api/v1/registration/id/{id} (DELETE)
    - /api/v1/registration/name/{name} (DELETE)
  - Get all the existing registrations or a specific one
    - /api/v1/registration (GET)
    - /api/v1/registration/id/{id} (GET)
    - /api/v1/registration/name{name} (GET)
  - PING operation for general service availability
    - /api/vi/ping
- Export Distro has no REST APIs other than PING!!!

See https://wiki.edgexfoundry.org/display/FA/APIs--Export+Services--Client+Registration+API+Examples#APIs--ExportServices--ClientRegistrationAPIExamples-RegisterforJSONformatteddatatobesenttoMQTTtopic for some example registrations

Export Distro under the hood

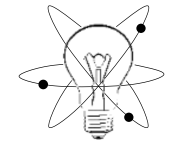
- Requires knowledge of EAI patterns and Spring Integration of the same
  - Each Core Data Event/Reading enters the system via the 0MQ “endpoint” or sink and becomes a message in the system
  - Each message has to be copied per client registration as a new message pushed back into the channel
  - Each message then goes through the EAI engine’s collection of message filter, router, formatter, transformation nodes – each node connected to the next in the chain via a message pipe.
  - At the end of the channel, the resulting message is pushed to an output endpoint (MQTT Topic, REST address, etc.)
- Allows for export distro to incorporate all sorts of filters, transformation, routing, and additional endpoint types in the future
  - Just add more Spring Integration beans into the pipe
  - Allows for alternate EAI implementations (e.g. Apache Camel)

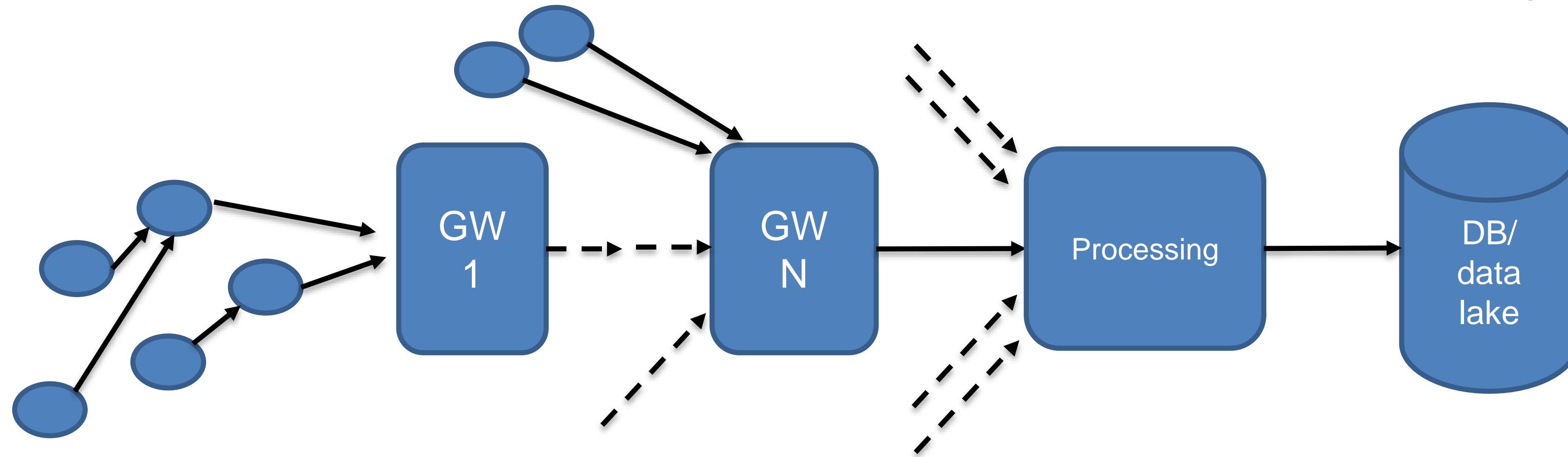
Export Distro EAI System



Data management for IOT intro

A General IOT system:





IoT edge devices

Layer 1 Gateway

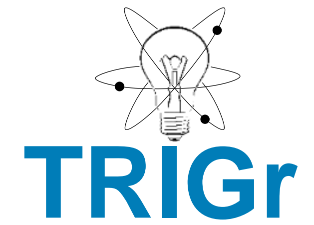
Layer N Gateway

Processing, Events & Analytics

Datacenter

DELL EMC

IOT system challenges:



- New edge devices are introduced and others retire constantly and the system.
- There is a huge amount of components from many types
- Geographical dispersion

These lead to the large amount of current management frameworks.
The management framework focus:
- management of the components themselves
- creation of analytics
- integration of the components.

DELLEMC

The missing piece – Data management:

- How to store the data at different levels of the system?
- How to protect the data against logical corruption or physical disaster at different levels of the system?
- What happens when storage capacity runs out at different levels of the system?
- What happens when bandwidth is limited or communications are disrupted?
- How can all the above be managed in systems that dynamically change all the time?
- How can you track and control cost of storage?

DELL EMC

How to store the data at different levels of the system

TRIGr
TECHNOLOGY RESEARCH INNOVATION

Data is intercepted by sensors

- Some sensors have local storage
  - Which data should be stored
  - Which data should be discarded in case of connectivity problems
- Some sensors have processing capabilities
  - Data can be preprocessed and only processing results can be sent by the sensor to the gateway

Data is sent from sensors to gateways, which have more storage and more processing capabilities:

- Same questions are relevant for the gateways

From the gateways data can be sent to local data centers or the clous.

DELLEMC

How to protect the data against logical corruption or physical disaster

TRIGr
TECHNOLOGY RESEARCH INNOVATION

- At each level of the IOT system there are different amounts of data and difference importance of the data
- At the sensors there is very limited storage and large amount of data generated, and thus data is usually discarded very fast is it cannot be transmitted
- At Gateways, data should be cached, not all data can be transferred to the cloud
  - Do we need to keep other copies of the data in different nearby gatways?

- At cloud level, data is stored on data bases and streams, these mechanisms have data protection support, but it needs to be managed.

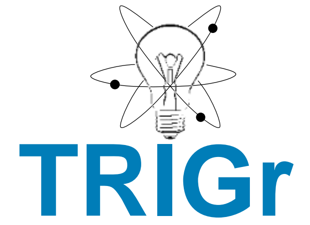
What happens when storage capacity runs out at different levels of the system

If data cannot be transferred and storage is full some data should be discarded

- Down-sampling
- Low-pass filters
- Lower bits per sample
- Approximations and adaptations (like PCM vs ADPCM)
- Image filters and/or resolution reductions (JPEG, JPEG2K, MPEG)
- Video stream frame rate reduction
- Color sample resolution
- Transformations like CELP family of compression for voice specific data
- Data Stripping
- Feature extraction
- Do nothing at all

DELLEMC

What happens when bandwidth is limited or communications are disrupted?



- System can store data locally until communication is back
- But storage gets full and then data needs to be discarded or processed
- There has to be a policy to control which data is discarded and which data

DELLEMC

How can you track and control cost of storage

The cloud theoretical has infinite amount of storage

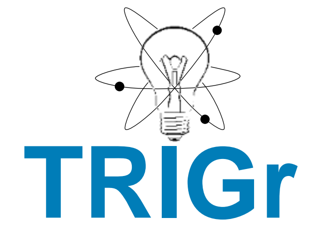
- But storage costs
- There has to be a mechanism to control and manage the cloud data
    - Data retention
    - Calculate cost
    - Calculate which data can be discarded or has its resolution reduces

At the edge

- Do we have enough storage at the edge?
- Do we have to much storage at the edge
- Smart calculation of edge storage can reduce and help controlling the cost.

DELLEMC

Cross IOT components data management



- Create one holistic entity and define a system-wide data management policy and governance entity.
- The same policy, or parts of it, can be employed on the backend and on any of the components in the system.

- here are tradeoffs between compute, storage and bandwidth that need to be clearly modeled, and the resulting policies and system behaviors must be built accordingly.

DELL EMC

Data management for EdgeXFoundry

DELL EMC

Data management challenges CORE

- Data arrives to core, there is very little data management available
  1. Data is thrown from DB when DB is full
  2. There is no policy for controlling the sensors – they can create overflow
  3. There is no option to filter the data once it entered the DB
  4. There is no option to create smart data reducrtion techniques to free space in the DB.
  5. Data is transmitted to the export services via the message queue immediately.

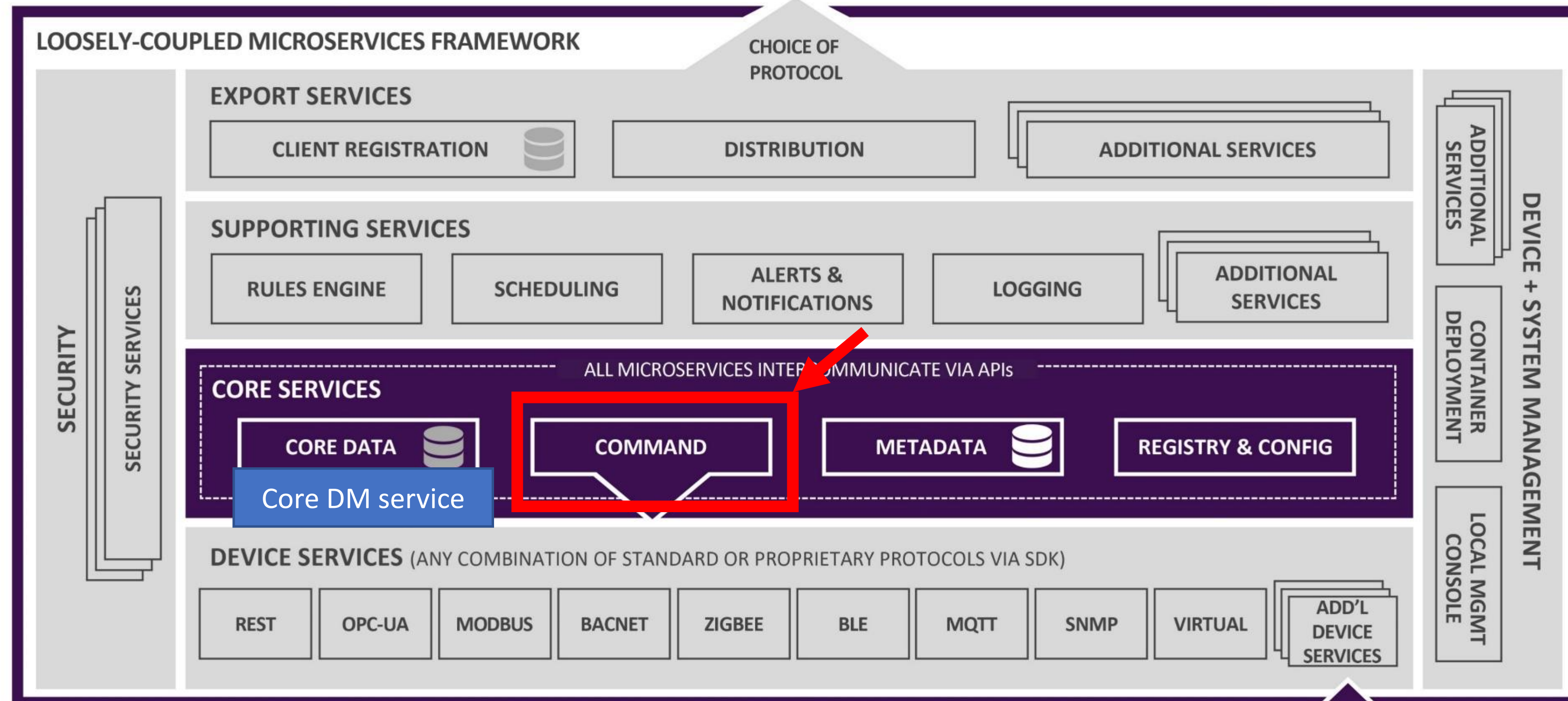
Potential solution for Core data management

- We can create a core data management service
- The data management solution will implement the same API of core data and extend it for policy management
- Each device service may be able to register to the core data management
- For a device service registered to the core data management there all API call will go through the data management service and not through the core data service.

Potential solution for Core data management cont.

- The core data management service will implement a policy engine
- The policy configuration can be kept in the meta data service.
- One option is to add to attach a policy to a device profile
- The policy engine will make the following decision:
  - Filter incoming data from the core service in case of congestion
  - Notify devices to stop sending data or continue (though the device service)
    - For smarter devices the policy engine can notify a transmission policy.
  - Periodically perform data base wide data reduction operation
    - Delete non relevant data
    - Reduce resolution for older data
    - Run smart operators on data inside the data base

Big Picture and Command



Data management challenges External services

- The external services do have some primitive data services
- for export distro to incorporate all sorts of filters, transformation, routing, and additional endpoint types in the future
  1. Transofrmation options are very limited today
  2. There is no operation on historical data just on current data
  3. There is no bandwidth management
  4. There is very limited failure management (WAN failure)

Potential solution for external services data management

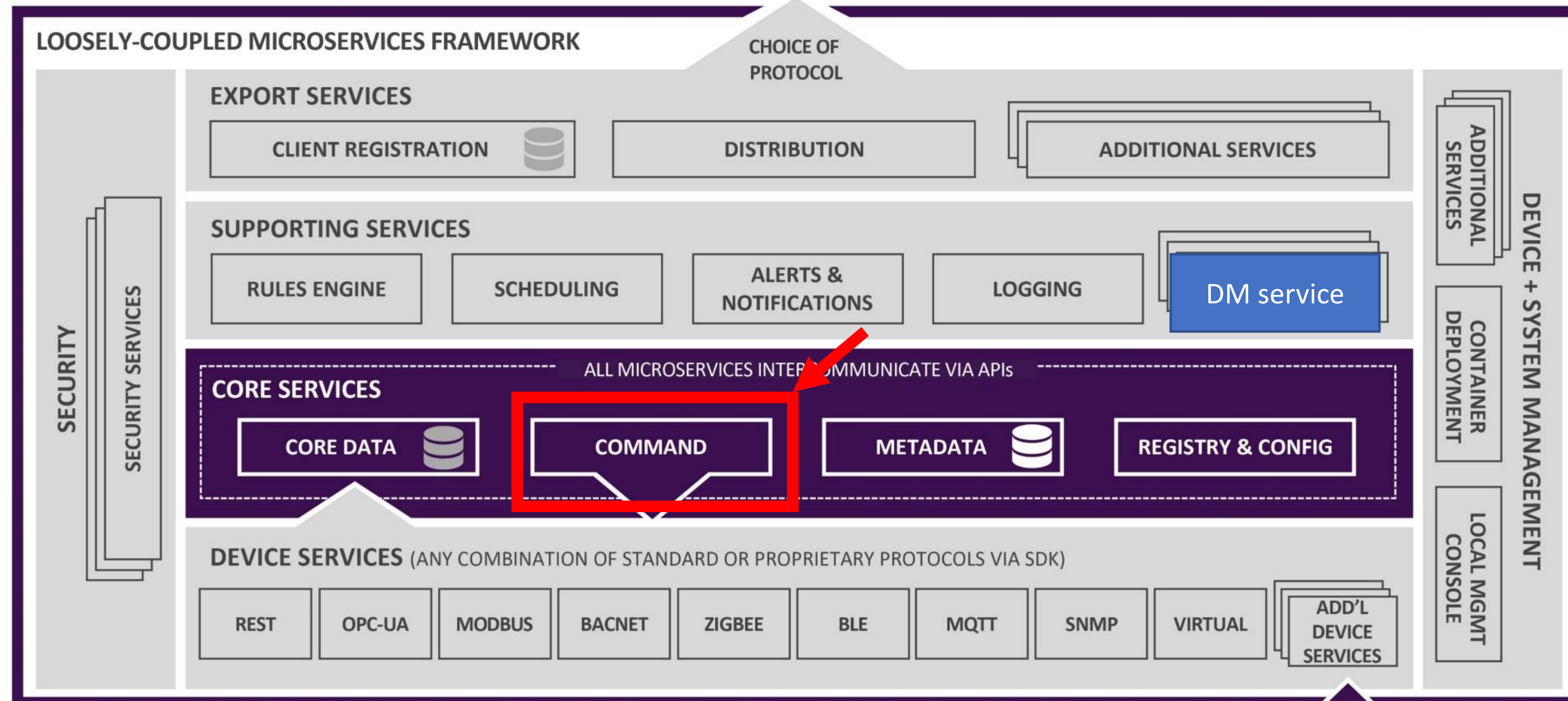
- The data management for external services can be a completely separated service
  - The service will register to the message queue and receive all messages
  - The data management service will have a policy engine, which will allow configuration of different policies on a per device, per device type and per location and other criteria.
  - The policy will include
    - retention policy for the data
    - Operation on the data when retention is reached (i.e. reduce resolution etc..)
    - Failure handling policy (what happen when network fails or when storage is to slow)
    - Data transmission policy (to next layer)

Potential solution for external services data management

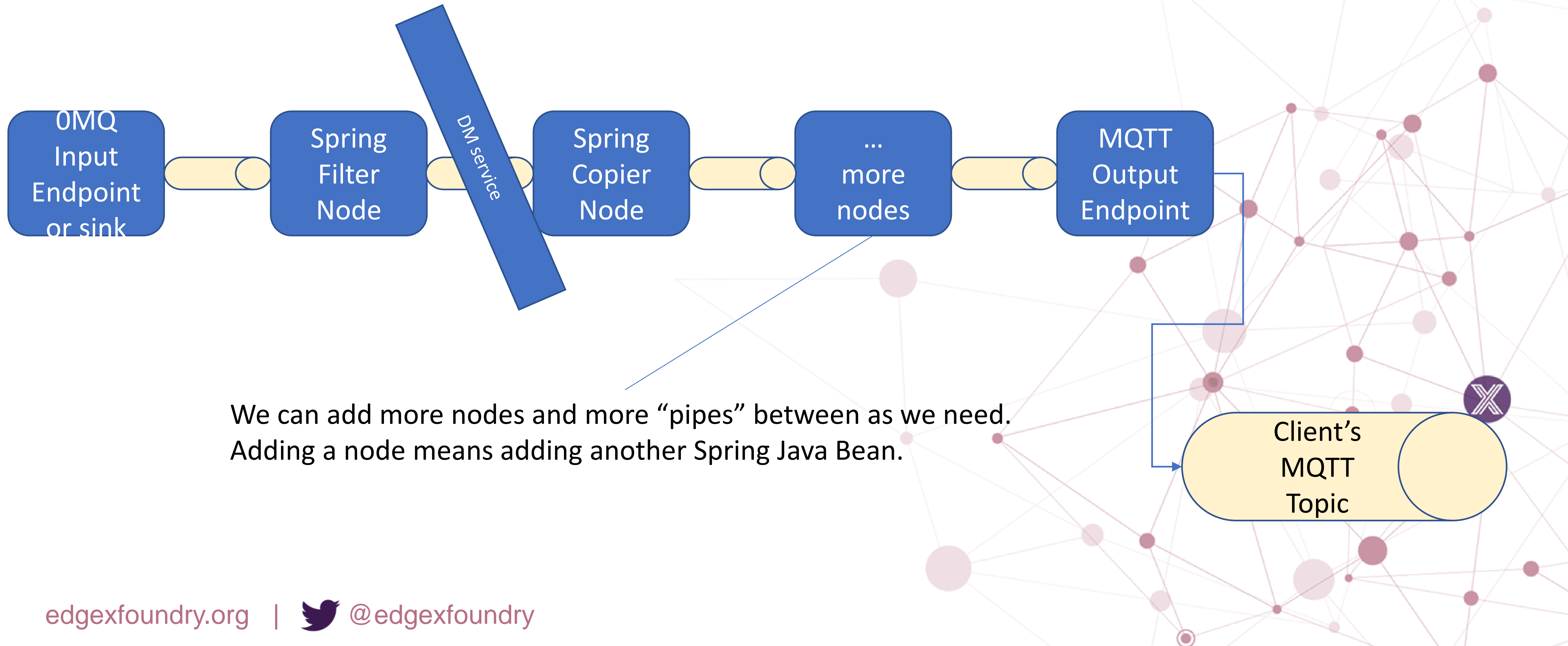
- The data arriving to the external services will be filtered according to the policy and kept in a data base
- Periodically the service will create a cleanup on the data base and apply the data retention and data management policies on data inside the data base.
- The data management service will publish data to another message ques based on the transmission policy
  - i.e data does no necessarily sent immediately as arrive
  - Based on available bandwidth and policy send the relevant data to other registered edge devices

EDGE X FOUNDRY

Big Picture and Command



Export Distro EAI System



We can add more nodes and more "pipes" between as we need.
Adding a node means adding another Spring Java Bean.

Data management suggestions

May the demo gods be with us