에지컴퓨팅 오픈소스 EdgeX 소개

홍 정 하
ETRI 지능형IoE네트워크연구실
jhong@etri.re.kr

(참고) EdgeX에 관한 자료는 EdgeX Foundry (https://edgexfoundry.org) 자료를 인용하였습니다.

목차

- IoT Edge Computing

- 오픈소스 EdgeX 소개
  - EdgeX Foundry 프로젝트 개요
  - EdgeX Architecture
  - EdgeX API Walk Through 데모 개요

- Appendix
  - EdgeX 설치 및 설정 방법

Buzzwords

- Edge gateway
- Edge computing
- Fog computing
- Intelligent Edge computing
- Edge cloud
- …

Microsoft Azure

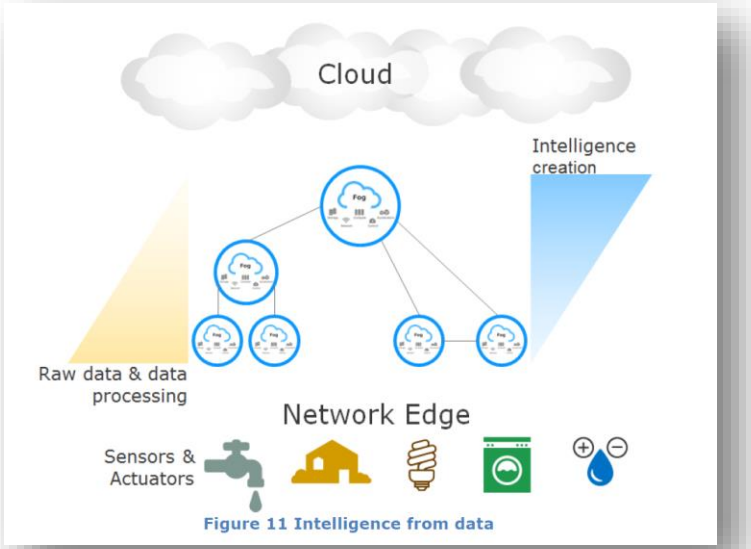
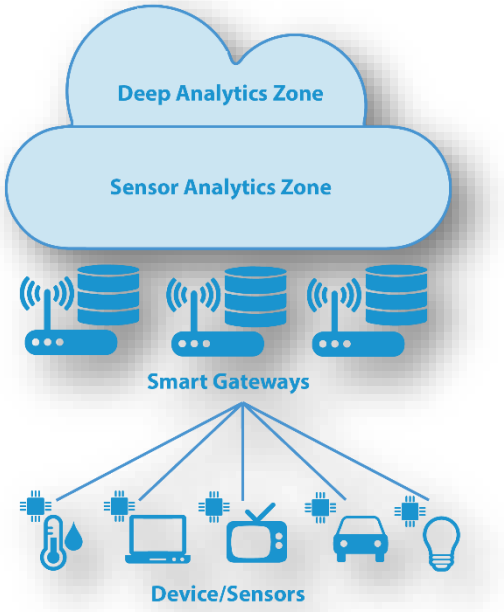
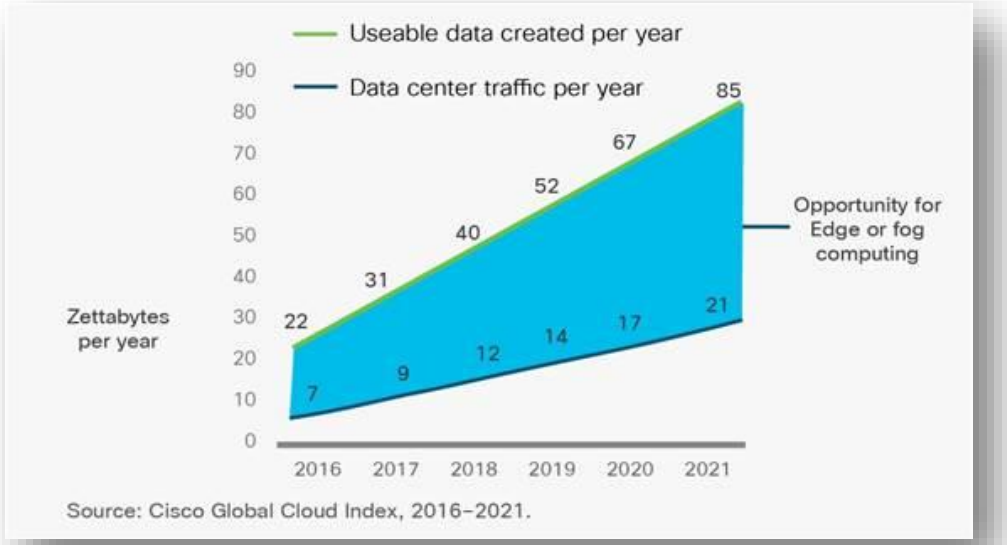


Figure 11 Intelligence from data



제타바이트 시대

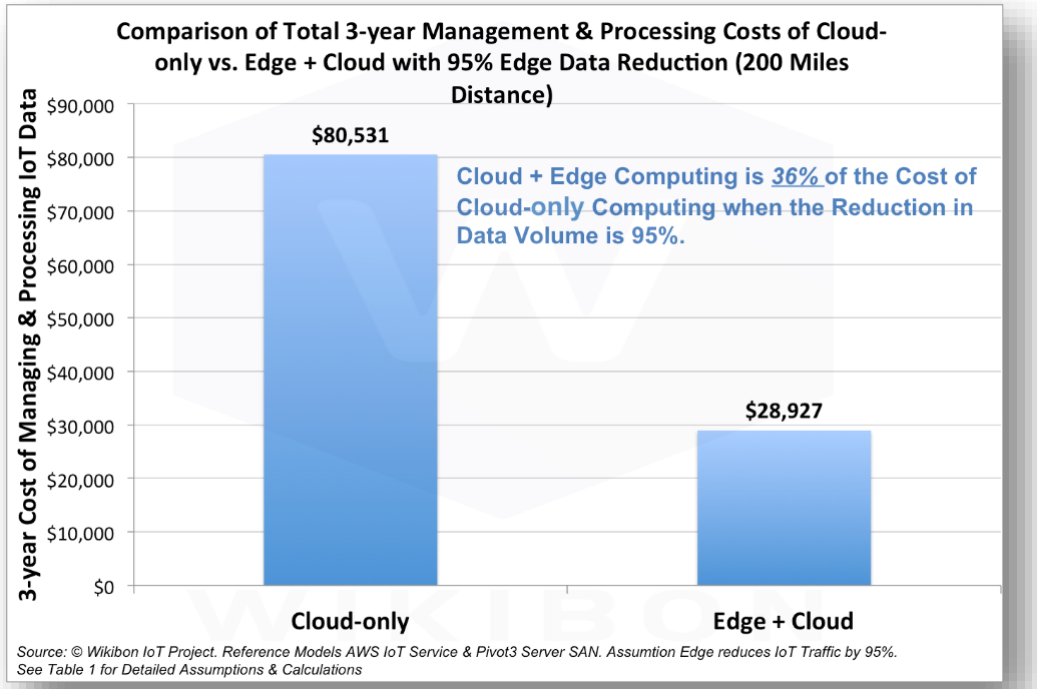
- Cisco GCI estimates that nearly 850 ZB will be generated by all people, machines, and things by 2021, up from 220 ZB generated in 2016
- Much of this ephemeral data is not useful to save, but we estimate that approximately 10 percent is useful (85 ZB)
- Useful data also exceeds data center traffic (21 ZB per year) by a factor of four
- Edge or fog computing might help bridge this gap

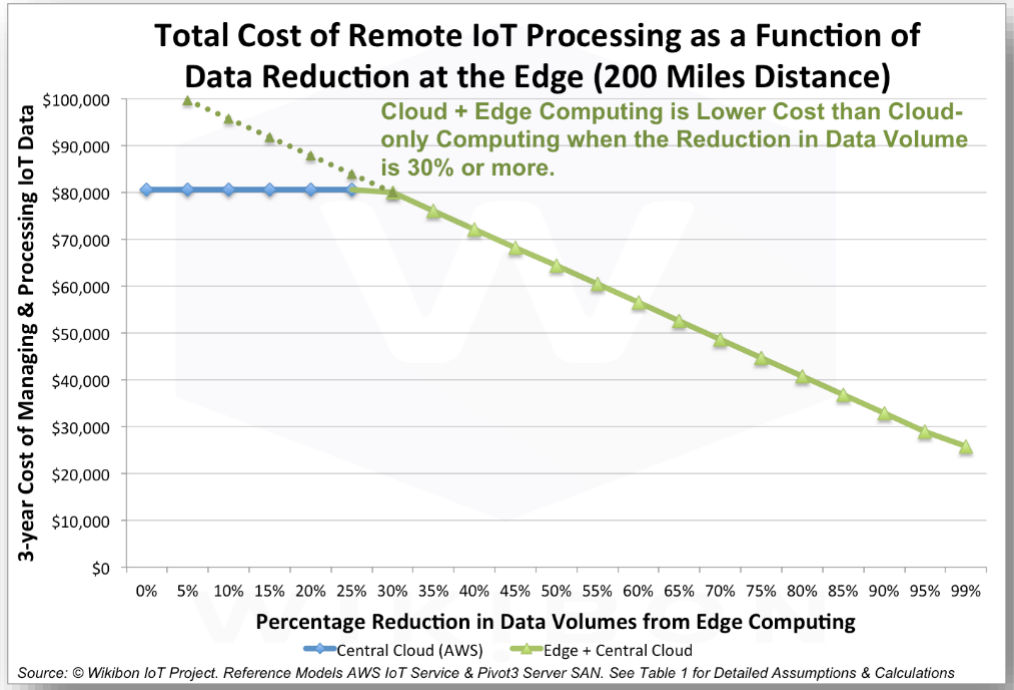


출처: Cisco Global Cloud Index: Forecast and Methodology, 2016–2021 White Paper

Edge and Cloud computing

- IoT management architecture
  - Changes Cloud-only to combining edge and cloud architecture



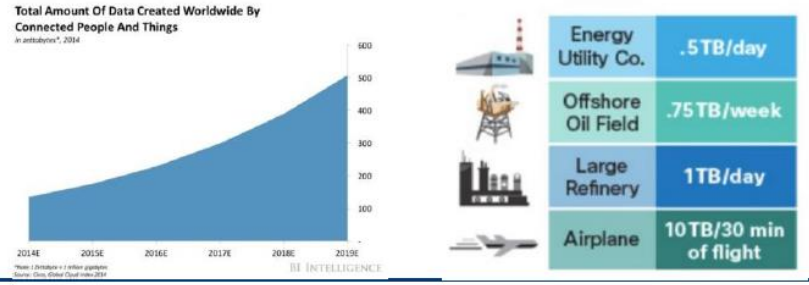


IoT Edge (1/3)

- At the verge of "4th Generation Industry Revolution" era, we need;
  - **Efficient Transmission / Analytics / Processing** of the explosively increasing IoT Data
  - **Real-time Data Analytics** for handling industry needs in Industrial IoT solutions

Increasing # of Connected Devices/Data

- **Explosive data transmission under BW limitation**
  - Amount of IoT Data in 2019 : 507 ZB[1)]
- **Orchestration among a various types of data**
  - Complexity of Cloud ↑ and Efficiency ↓



1) Projected by Cisco ('15), 1ZB=1 Trillion GB

Emerging Services of Time-Critical

- **Request real-time analytics : Safety/Efficiency**
  - Connected Car : Autonomous driving
  - Smart Factory : Abnormality and defects resolutions

[Source : 삼성전자 홍문기, EdgeX 기반 산업IoT용 Edge 컴퓨팅 기술 현황 및 전망, 2018.11]

IoT Edge (2/3)

Low Latency	Satisfy requirements from mission-critical apps (< a few tens of ms)
Data Locality	Prevent unnecessary exposure of privacy data as firewall for IoT data
Save Bandwidth	Save the required BW by IoT Data Analytic at Edge instead of Cloud
Disconnected Operation	Improve usability by Masking against disconnection of public network

Requirements	Cloud	Edge
Geo-distribution	Centralized	Distributed
Distance client and server	Multiple hops	One hop
Latency	High	Low
Delay Jitter	High	Very low
Location awareness	No	Yes
Support mobility	Limited	Supported
Location of service	Within the Internet	At the edge

[Source : 삼성전자 홍문기, EdgeX 기반 산업IoT용 Edge 컴퓨팅 기술 현황 및 전망, 2018.11]

IoT Edge (3/3)

- IoT Edge 플랫폼 기능 분류

Category	Functionality	Target Device
Collecting	Machine / Sensor Data Collection (Protocol – OPC UA, PLC, etc.) Stream Data Query (Data Redundancy Mitigation / Filtering, etc.) Status Monitoring – Machine / Sensor Connection	Raspberry Pi
Computing	Data Pre-Processing / Analytics Framework (TensorFlow, Caffe2, Flink, etc.) Running ML Engine : Vision / Voice Recognition (Learning at Cloud) Machine / Sensor Operation Control by Data Processing Result at Edge	PC, Mobile, etc.
Storage	Storage / compression / offering of private data Metadata management for data searching service Cyclic data backup and synchronization (Transferring data into cloud)	Server, NAS
Common	Service deployment / update / monitoring (Auto-scaling) Security : Authentication, Identification, Secure Communication	-

[Source : 삼성전자 홍문기, EdgeX 기반 산업IoT용 Edge 컴퓨팅 기술 현황 및 전망, 2018.11]

에지컴퓨팅 국외 기술 동향

- Cloud platform 기반 에지컴퓨팅 기술
  - AWS
  - Microsoft
  - Google
- IIoT 플랫폼을 위한 에지컴퓨팅 기술
  - GE Predix
  - Cisco IOx
- S/W 플랫폼 기술 (오픈소스)
  - **EdgeX Foundry**
- H/W 플랫폼
  - Nvidia
  - Intel
  - Google Edge TPU

EdgeX Foundry

(참고) EdgeX에 관한 자료는 EdgeX Foundry (https://edgexfoundry.org) 자료를 인용하였습니다.

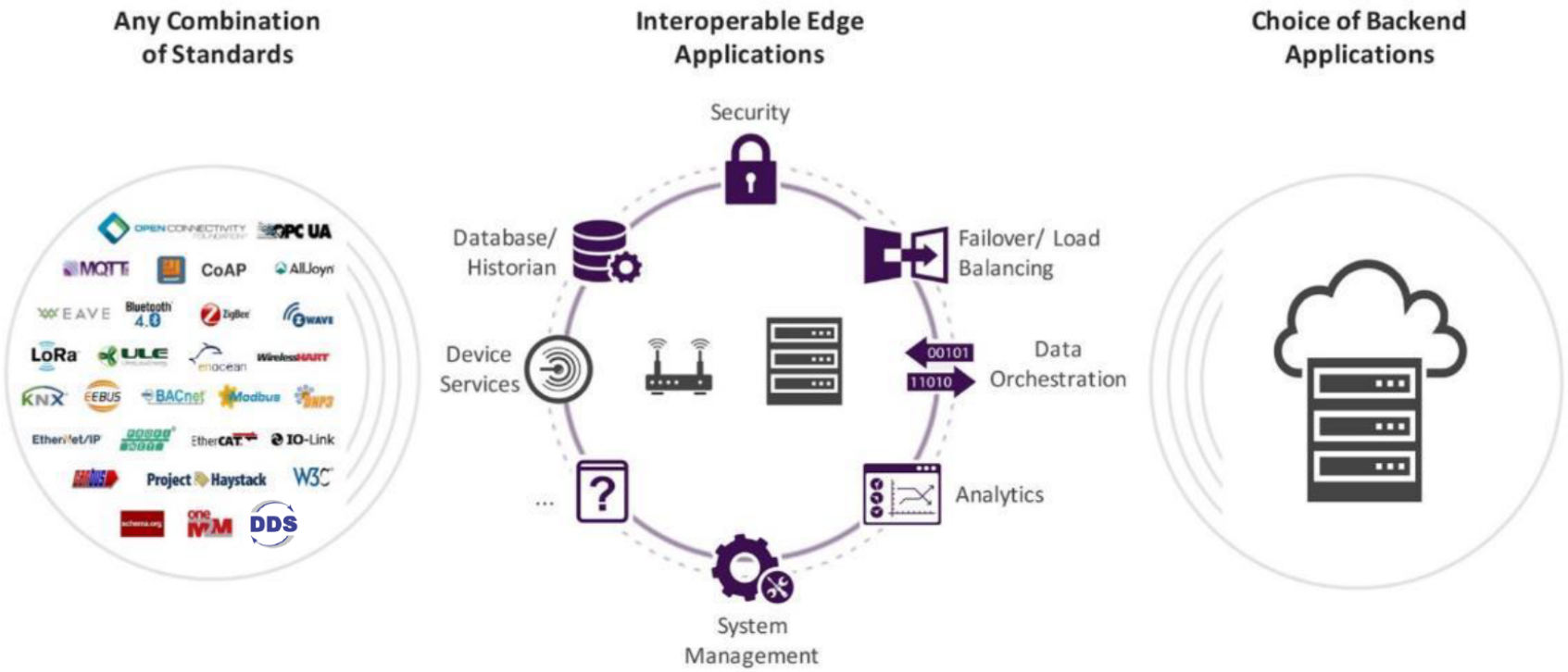
EdgeX Foundry

- Linux Foundation, Apache 2 project
- An open source, vendor neutral project (and ecosystem)
- A micro service, loosely coupled software framework for IoT edge computing
- Hardware and OS agnostic
- Focused on industrial IoT

EdgeX Foundry 목표

- Build and promote EdgeX as the common open platform unifying edge computing
- Enable and encourage the rapidly growing community of IoT solutions providers to create an ecosystem of interoperable plug-and-play components
- Certify EdgeX components to ensure interoperability and compatibility
- Provide tools to quickly create EdgeX-based IoT edge solutions
- Collaborate with relevant open source projects, standards groups, and industry alliances to ensure consistency and interoperability across the IoT

Bridging Standards with an Ecosystem of Apps



The EdgeX framework enables developers to **decouple their preferred Edges and Clouds** as close to the physical world as possible, **minimizing reinvention of table stakes elements** for data ingestion, security and management and benefitting from **flexibility and choice** (e.g. build vs. buy from the open ecosystem).

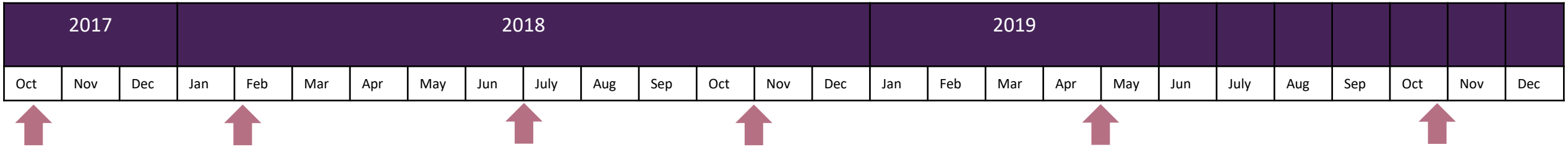
[Source : 삼성전자 홍문기, EdgeX 기반 산업IoT용 Edge 컴퓨팅 기술 현황 및 전망, 2018.11]

EdgeX History (1/2)

- Chartered by Dell IoT marketing in July 2015
  - A Dell Client CTO incubation project (Project Fuse)
- Designed to meet interoperable and connectivity concerns at the IoT edge
- Started with over 125,000 lines of Dell code
- Entered into open source through the Linux Foundation on April 24, 2017
  - Started with nearly 50 founding member organizations; today we have more than 75

EdgeX History (2/2)

- Release Cadence: 2 formal releases a year



'Barcelona' Release

(Released Oct 20 2017)

- Improved fit and finish, formalized Core Service APIs, additional Device and Export Services, test apparatus

'California' Preview'

(Made available Jan 2018)

- Drop-in Go Lang micro service replacements demonstrating reduced footprint and higher performance

'California' Release

(Released June 2018)

- First integration of security
- Run in < 256MB RAM, come up in < 10 sec

'Delhi' Release

(Oct 2018)

- Additional security and first manageability capabilities
- Go / C device service SDKs
- Sample device services (e.g. Modbus, MQTT, OPC-UA)
- EdgeX UI

'Edinburgh' Release

(Apr 2019)

- Certification Program
- Improved and more scalable northbound connectors / application services
- Additional southbound connectors to common protocols and devices
- ARM 32 support

'Fuji' Release

(Oct 2019)

- Load balancing
- Multi-host EdgeX
- Additional security and system management capability

75+ 회원들

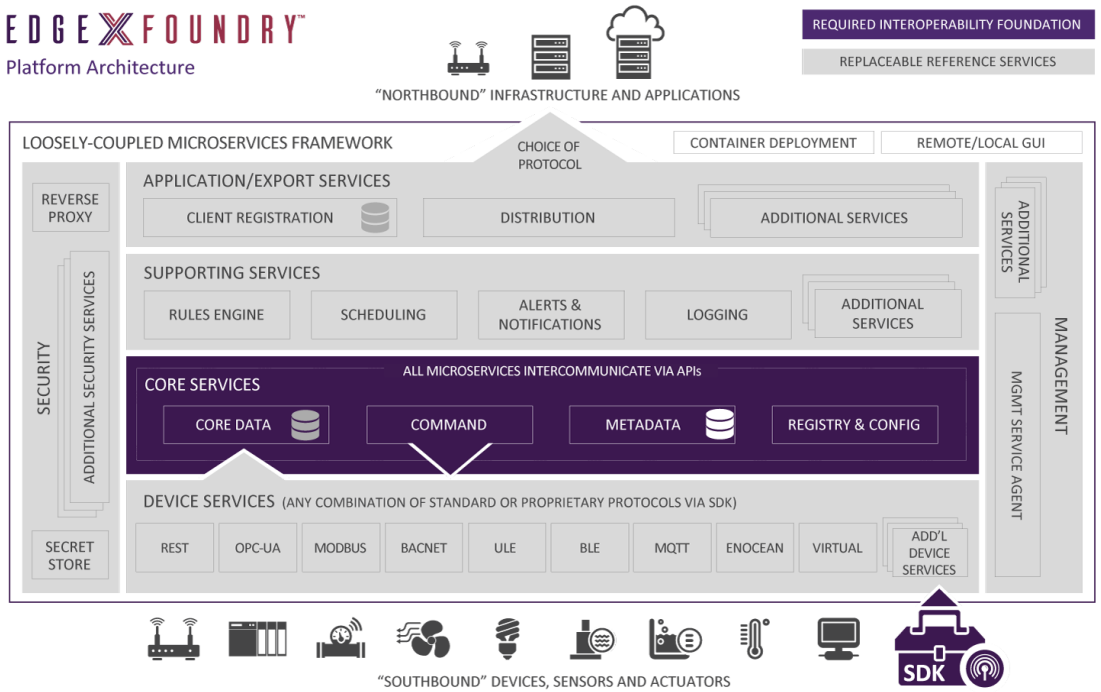

ABSOLUTE
aicas realtime
Alleantia
AMD
ANALOG DEVICES
AXZON
BASKING
Beechwoods
ubuntu Delivered by Canonical
CAVIUM
CertusNet
CIVIL INFRASTRUCTURE PLATFORM
CLOUD FOUNDRY
CloudPlugs Connecting Things
Cumulocity A SOFTWARE AG COMPANY
DELL EMC
DEVICE AUTHORITY Trust for every Thing
enigmedia
enocean
EPISENSOR
ETRI Electronics and Telecommunications Research Institute
FIWARE FOUNDATION
FOGHORN
FORGEROCK
IMS EVOLVE
industrial internet CONSORTIUM
inwinSTACK G.O.D. Great OpenStack Delivery
IOTech
IoTium www.iotium.io
Irish Manufacturing Research
ISSAT Mateur
KMC CONTROLS
Linaro
MAINFLUX Internet of Things Solutions
MOCANA
modius
NETFOUNDRY SPIN UP YOUR NETWORK
neustar
OMG OBJECT MANAGEMENT GROUP
OPTO 22
Project Haystack
Queen's University Belfast
redislabs home of redis
REPLY CONCEPT
RIOT
Rubicon
SAMSUNG
SAMSUNG SDS
Sixgill Govern IoE
striim
SWITCH AUTOMATION
THALES
twobulls
ULE ALLIANCE
V5 SYSTEMS
vmware
Volterra EDGE SERVICES
萬向集團 WANXIANG GROUP
xage SECURITY
ZEDEDA
Zephyr
ZingBox

EdgeX Architecture

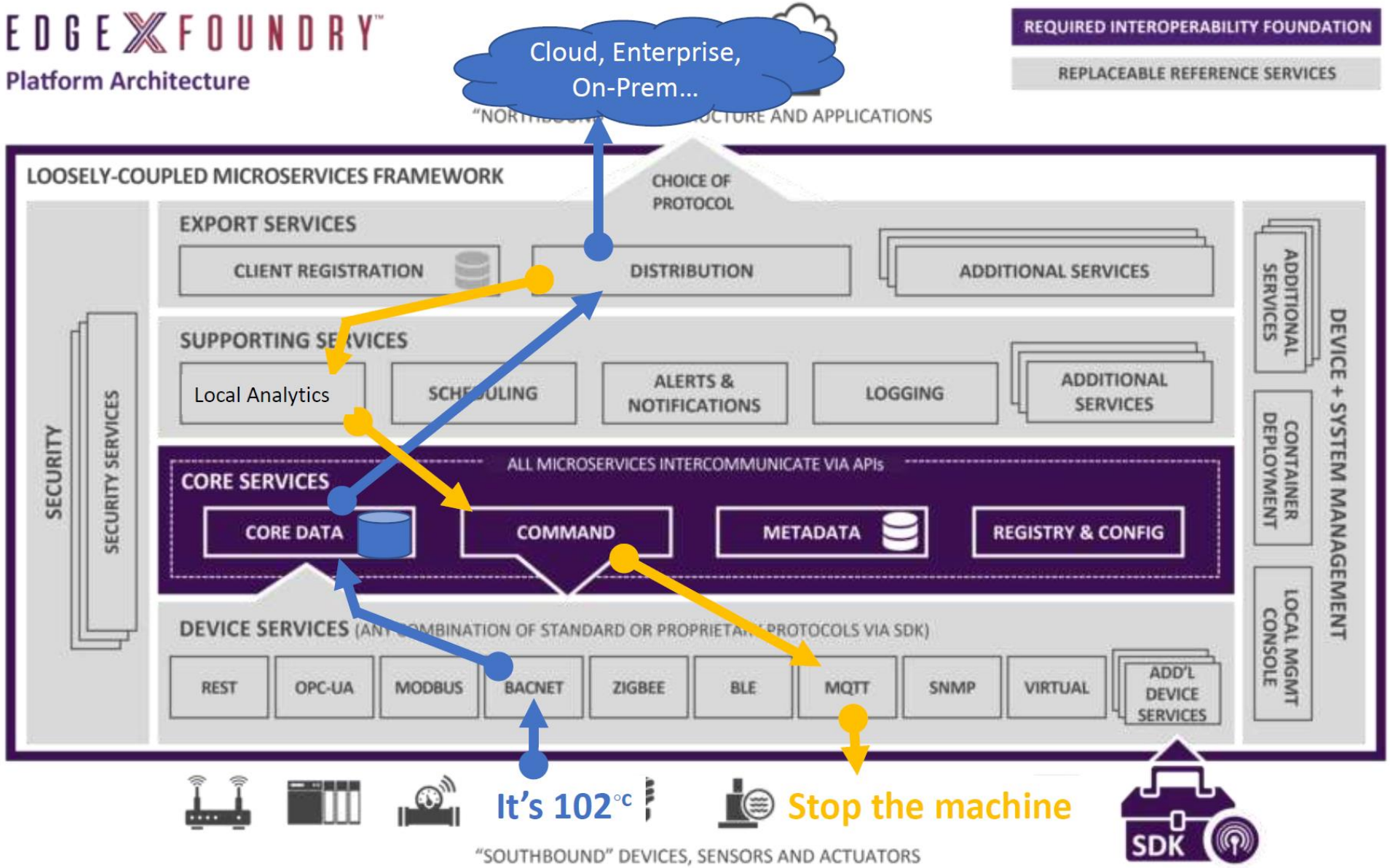
(참고) EdgeX에 관한 자료는 EdgeX Foundry (https://edgexfoundry.org) 자료를 인용하였습니다.

EdgeX 동작 개요

- A collection of a dozen+ micro services
  - Written in multiple languages (Java, Go, C, …)
- EdgeX data flow:
  - Sensor data is collected by a **Device Service** from a thing
  - Data is passed to the **Core Services** for local persistence
  - Data is then passed to **Export Services** for transformation, formatting, filtering and can then be sent “north” to enterprise/cloud systems
  - Data is then available for edge analysis and can trigger device actuation through Command service
  - Many other services provide the supporting capability that drives this flow



- REST communications between the service
  - Some services exchange data via message bus (core data to export services and rules engine)
- Micro services are deployed via Docker and Docker Compose



EdgeX 구조적 원리

- EdgeX Foundry must be **platform agnostic** with regard to hardware, OS, distribution/deployment, protocols/sensors
- EdgeX Foundry must be **extremely flexible**
  - Any part of the platform may be upgraded, replaced or augmented by other micro services or software components
  - Allow services to scale up and down based on device capability and use case
- EdgeX Foundry should provide “reference implementation” services but **encourages best of breed solutions**
- EdgeX Foundry must provide for **store and forward** capability (to support disconnected/remote edge systems)
- EdgeX Foundry must support and **facilitate “intelligence” moving closer to the edge** in order to address
  - Actuation latency concerns
  - Bandwidth and storage concerns
  - Operating remotely concerns
- EdgeX Foundry must **support brown and green device/sensor** field deployments
- EdgeX Foundry **must be secure and easily managed**

EdgeX 마이크로 서비스 계층

- Contextually, EdgeX micro services are divided into 4 layers
- Crudely speaking, the layers of EdgeX provide a dual transformation engine
  - 1x - Translating information coming from sensors and devices via hundreds of protocols and thousands of formats into EdgeX
  - 2x - Delivering data to applications, enterprises and cloud systems over TCP/IP based protocols in formats and structures of customer choice

EdgeX 기술

- A majority of the micro services are written in Go Lang
  - Previously written in Java
  - Some Device Services written in C/C++
  - A user interface is provided in JavaScript
  - Polyglot belief – use the language and tools that make sense for each service
- Each service has a REST API for communicating with it

- Uses MongoDB to persist sensor data at the edge
  - Also stores application relevant information
  - Allows for alternate persistence storage (and has been done in the past)

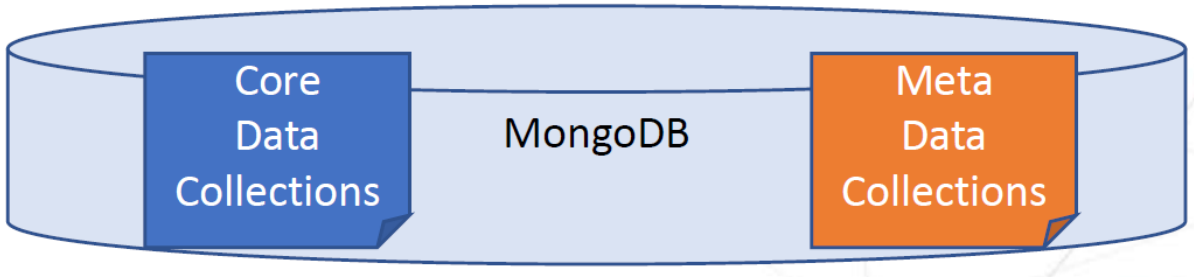
- A message pipe connects Core Data to Export Services and/or Rules Engine
  - Uses ZeroMQ by default
  - Allow use of MQTT as alternate if broker is provided

- Uses open source technology where appropriate
  - Ex: Consul for configuration/registry, Kong for reverse proxy, Drools for rules engine,…

Core Services

core data, metadata, command & configuration/registration

- Offers temporary persistence of edge data and facilitates actuation of things
  - Needed to support disconnected edge modes, latency concerns, costs of transport to the cloud
- Collects sensor data
- Understand what sensors/devices are connected how to communicate with them
- Provision facility for new sensors/devices and device services
- Manage device actuation requests to device services/devices
- Provide micro service registry
- Provide micro service configuration



Supporting Services

logging, scheduling, notifications, rules engine

- Normal software application duties plus “edge intelligence”
- Logging provides centralized EdgeX logging to location of choice
  - Log to file system, database (MongoDB), other…
- Scheduling allows any EdgeX service to put tasks on the clock
  - Clean out old sensor data, check for new sensors, etc.
- Notifications gives the ability for any EdgeX service to send an alert
  - Notify internal or external to EdgeX
  - Send via communication means of choice (email, SMS, etc.)
- Rules engine provides the means to watch sensor data and trigger local actuation as necessary
  - The rules engine is a place holder for any “edge analytics”
  - Drools wrapped engine today

Export Services

export client, export distribution

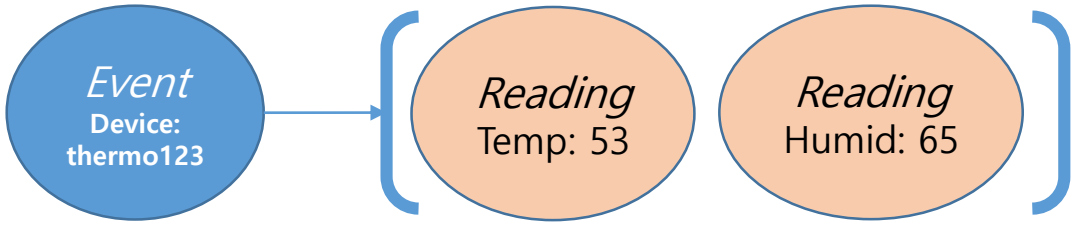
- Provides ability to get EdgeX sensor/device data to other external systems or other EdgeX services
  - External systems include cloud providers like Azure IoT Hub, Google IoT Cloud, etc.
- Export Client – allows for internal or external clients to:
  - Register for sensor/device data of interest
  - 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

Device Services

- A Device Service (DS) serves to translate the data produced and communicated by the thing into a common EdgeX data structure
- A DS also receives and translates generic commands from EdgeX and communicates that request to the devices for actuation in a language that the device understands.
  - For example, a DS may receive a request to turn off a Modbus PLC controlled motor. The DS would translate the generic EdgeX "shutoff" request into a Modbus serial command that the PLC/motor understands for actuation
- A device service may service one or a number of devices (sensor, actuator, etc.) at one time
  - A "device" that a DS manages could also be something other than a simple single physical device
  - It could also be another gateway (and all of its device), a device manager, or device aggregator that acts like a device or collection of devices to EdgeX
- The device service software developer kit (SDK) is a tool for generating the shell (the “scaffolding”) of a device service.
  - Initial SDK generates Java DS
  - It makes the creation of new device services easier
- The DS SDK & Meta Data Profile makes defining and provisioning new types of devices easier

Events & Readings

- EdgeX deals in Events and Readings
  - Collected by device services, persisted by core data, sent to cloud and other applications by export distro
- 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



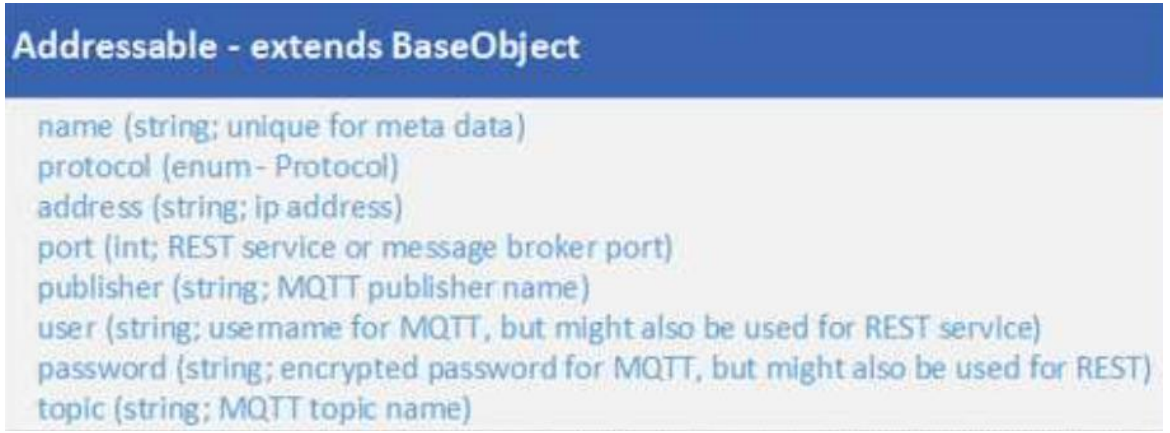
Value Descriptor

- 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"]}
```

Addressable

- An object that represents some way to address a device or service
  - Could be an HTTP / REST address / endpoint
  - Could be an MQTT topic
  - Could be a 0MQ topic
  - Could be a …
- Managed by Meta Data
  - Used by many micro services



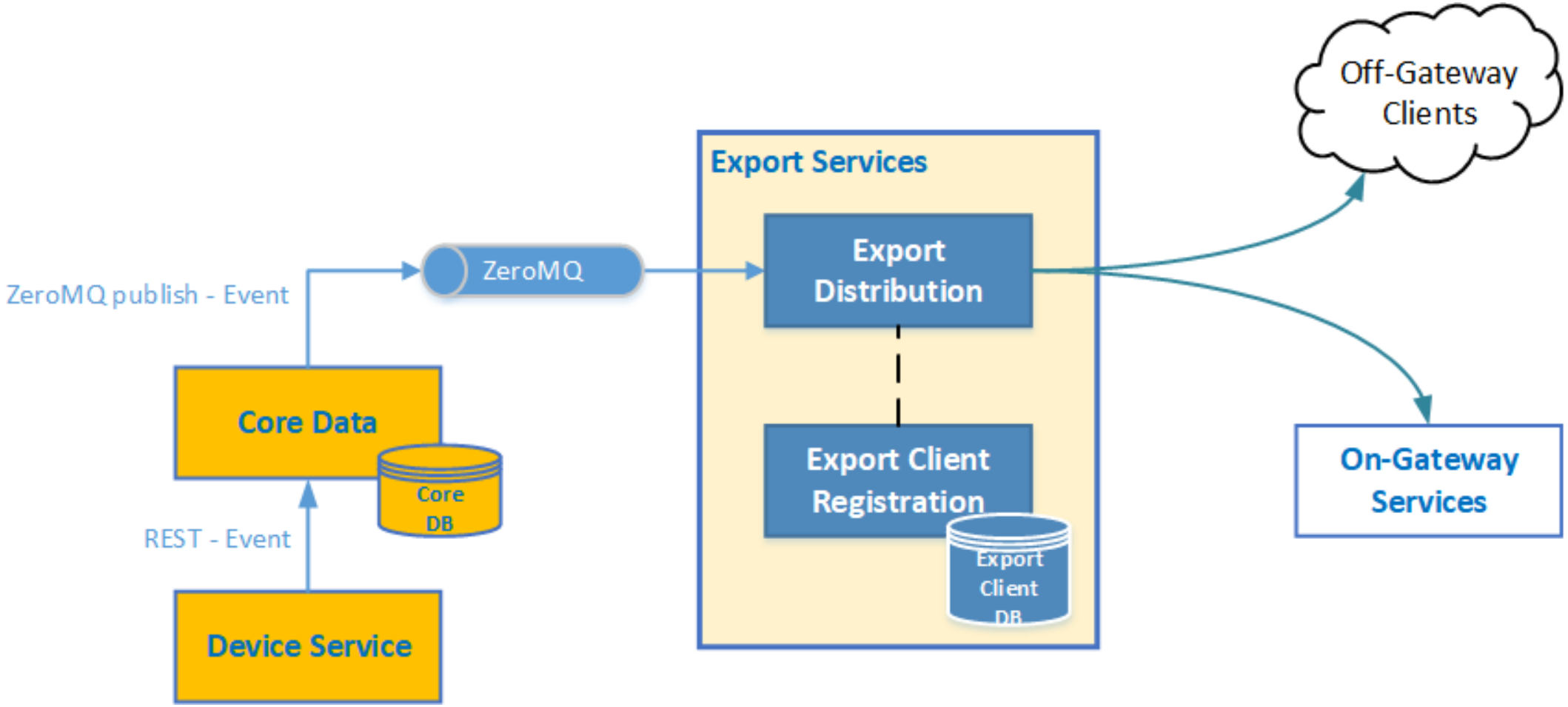
Device Profiles

- 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 BACnet thermostat device profile would provide general characteristics of thermostats
  - Specifically those communicating via the BACnet protocol
- It would describe the types of data a BACnet thermostat sends
  - Current temperature (as a float and in Celsius)
  - Current humidity (as a float and a percentage)
- It would describe what commands it responds to and how to send those commands
  - Get or Set the cooling set point (passing a float as a parameter)
  - Get or Set the heating set point (passing a float as a parameter)

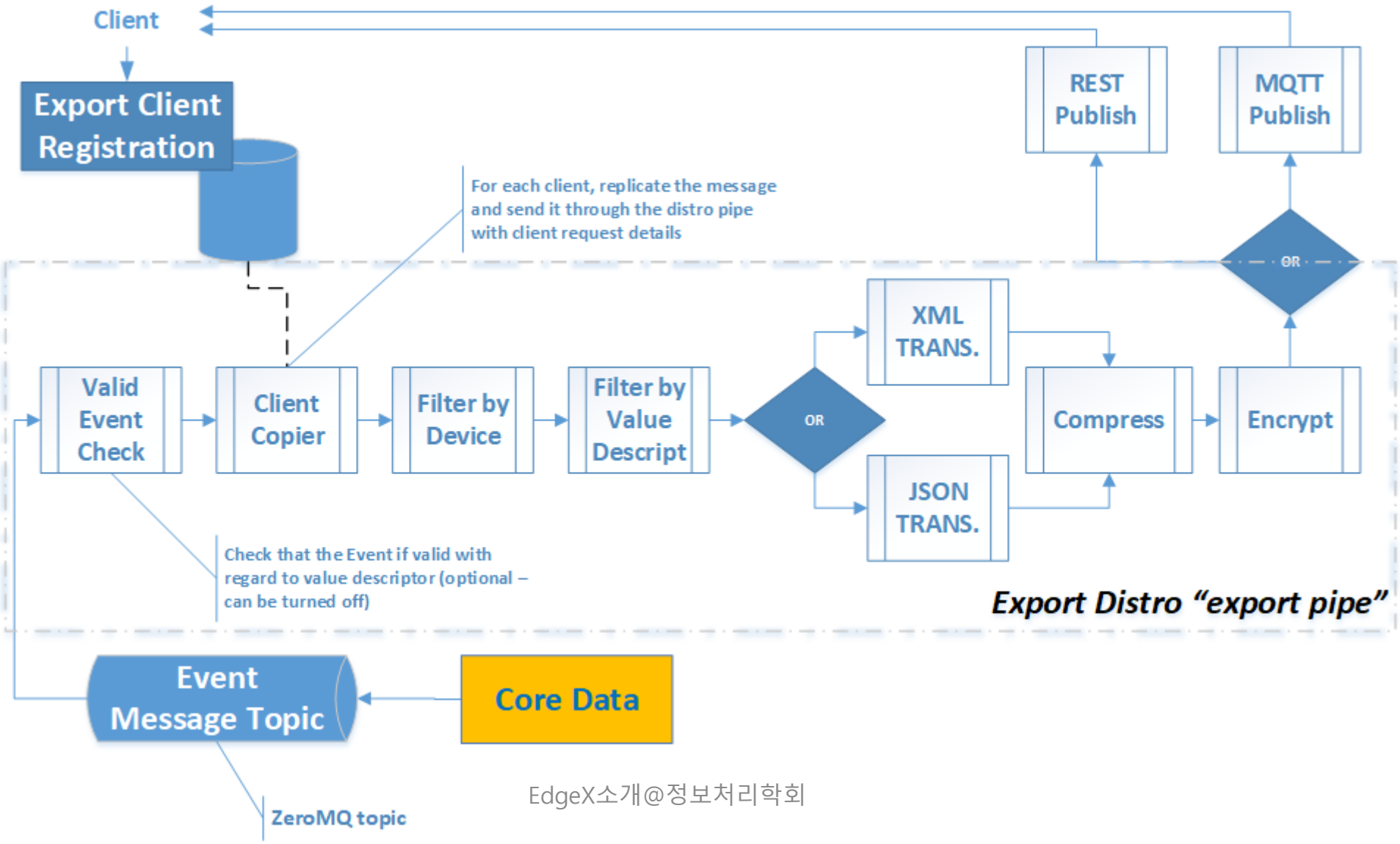
Export Distribution

- This is an Enterprise Application Integration engine
  - 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 Services “data flow”



Export Distro EAI Flow



EdgeX API Walk Through 데모

(참고) EdgeX에 관한 자료는 EdgeX Foundry (https://edgexfoundry.org) 자료를 인용하였습니다.

EdgeX API Walk Through 데모 목적

- This demonstration API walk through shows
  - how a device service and device are established in EdgeX
  - how data is sent flowing through the various services
  - how data is then shipped out of EdgeX to the cloud or enterprise system

- Through this demonstration
  - you will play the part of various EdgeX micro services by manually making REST calls (using Postman) in a way that mimics EdgeX system behavior

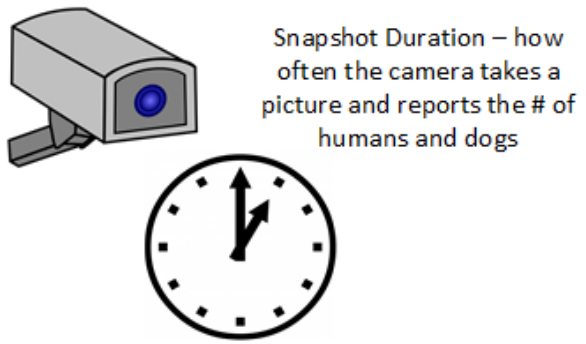
데모 Setup

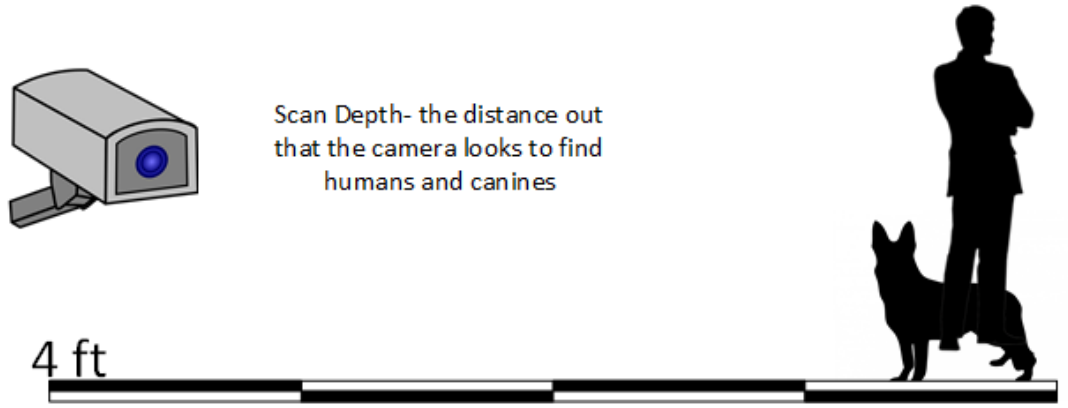
- Run all microservices but the virtual device service
  - Many of the API calls you make as part of this walk through are actually accomplished by the virtual device service (or any device service)

- Your manual call of the EdgeX APIs simulates the work
  - that a device service would do to get a new device setup and to send data to/through EdgeX

데모 시나리오 (1/2)

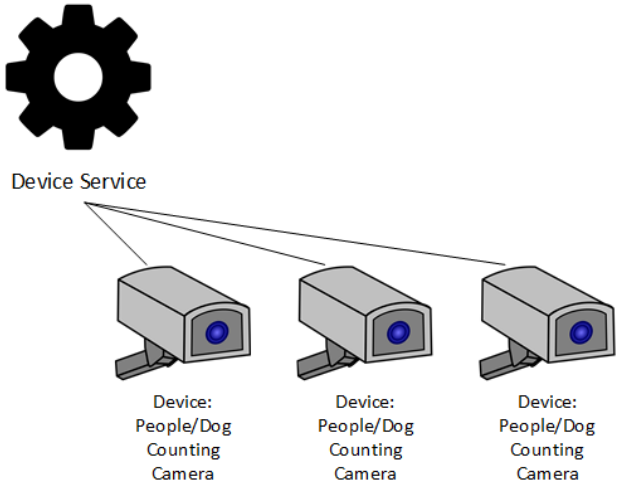
- 카메라 시나리오
  - 동작
    - 사진을 찍고 사진 속의 사람과 강아지의 수를 전송
  - 설정 파라미터
    - 사진 촬영 간격
    - Scan depth



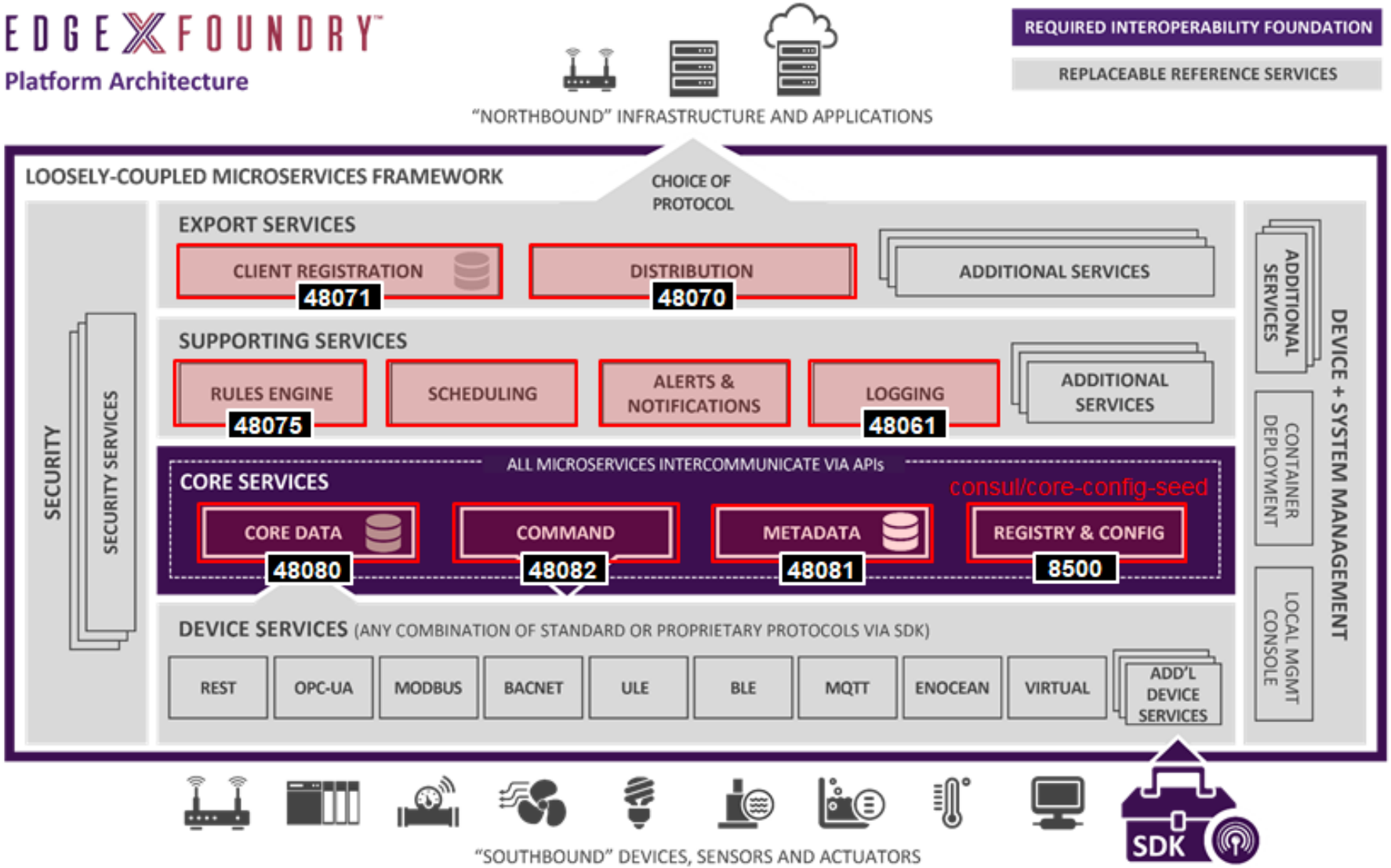


데모 시나리오 (2/2)

- 각 디바이스는 'Device Service'를 통해서 관리됨
  - Device Service는 디바이스(카메라)가 이해하는 프로토콜로 통신함
  - Device Service 디바이스로부터 데이터를 수집하여 EdgeX의 Core Data Service로 전달
    - In this case, the Device Service would be collecting the count of humans and dogs that the camera sees
  - Device Service는 EdgeX 및 외부의 request를 디바이스가 이해하는 프로토콜 request로 변환해줌
    - In this case, the Device Service would take requests to set the duration between snapshots and to set the scan depth and translate those requests into protocol commands that the camera understood



docker-edgex-volume
(edgex-files)



Device and Device Service Setup (1/6)

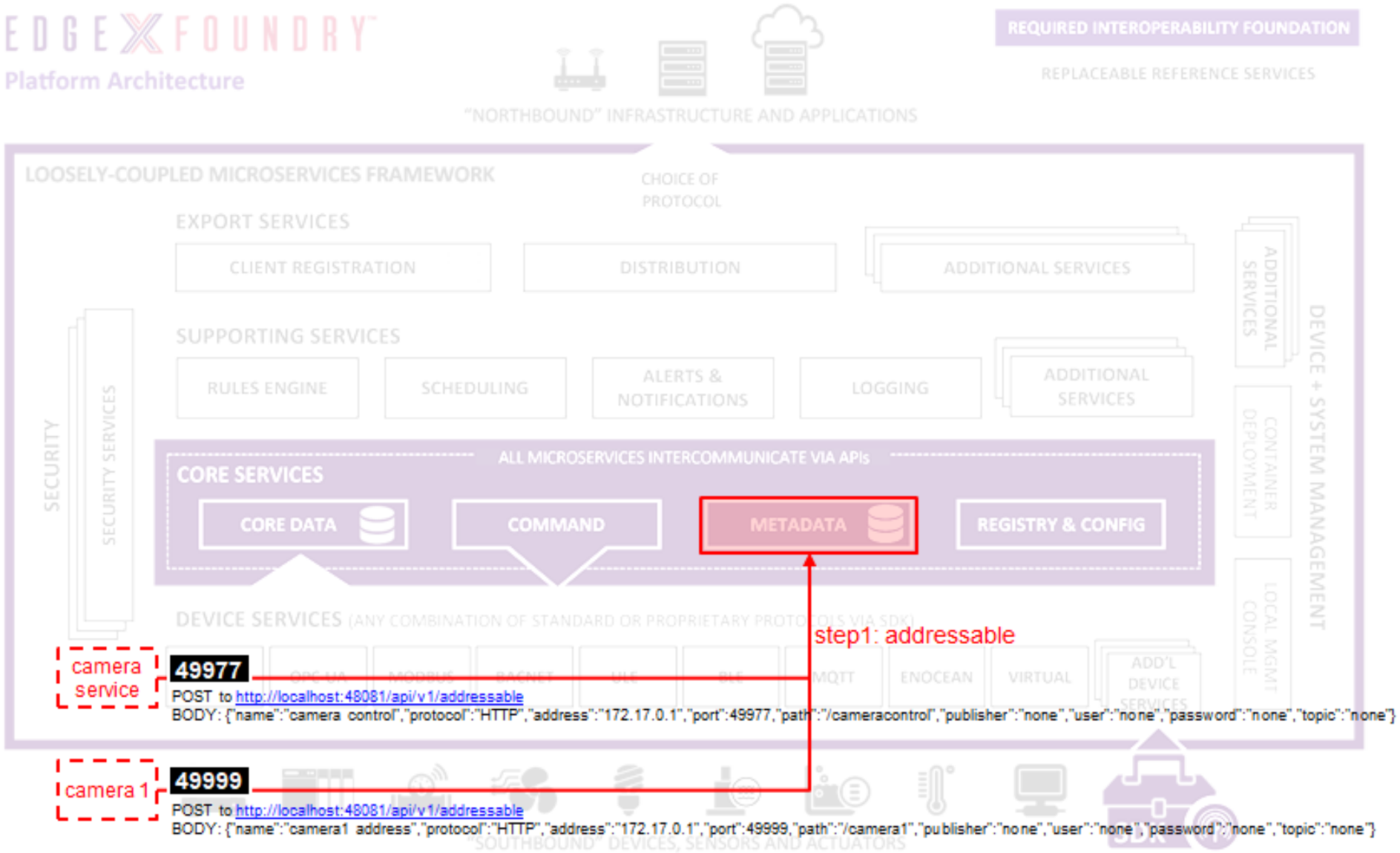
- Device Service tasks when it first starts can be categorized into
  - **Creating Reference Information in EdgeX**
    - Establish the reference information around the Device Service and Device.
  - **Creating the Device Service in EdgeX**
    - Make the Device Service itself known to the rest of EdgeX
  - **Provision a Device**
    - Provision the Devices that the Device Service will manage with EdgeX

- Reference information
  - Defining the address (called an Addressable) of the Device and Device Service
  - Establishing the new unit of measure (called a Value Descriptor in EdgeX) used by the Device

- After the initial start of a Device Service, these steps are not duplicated

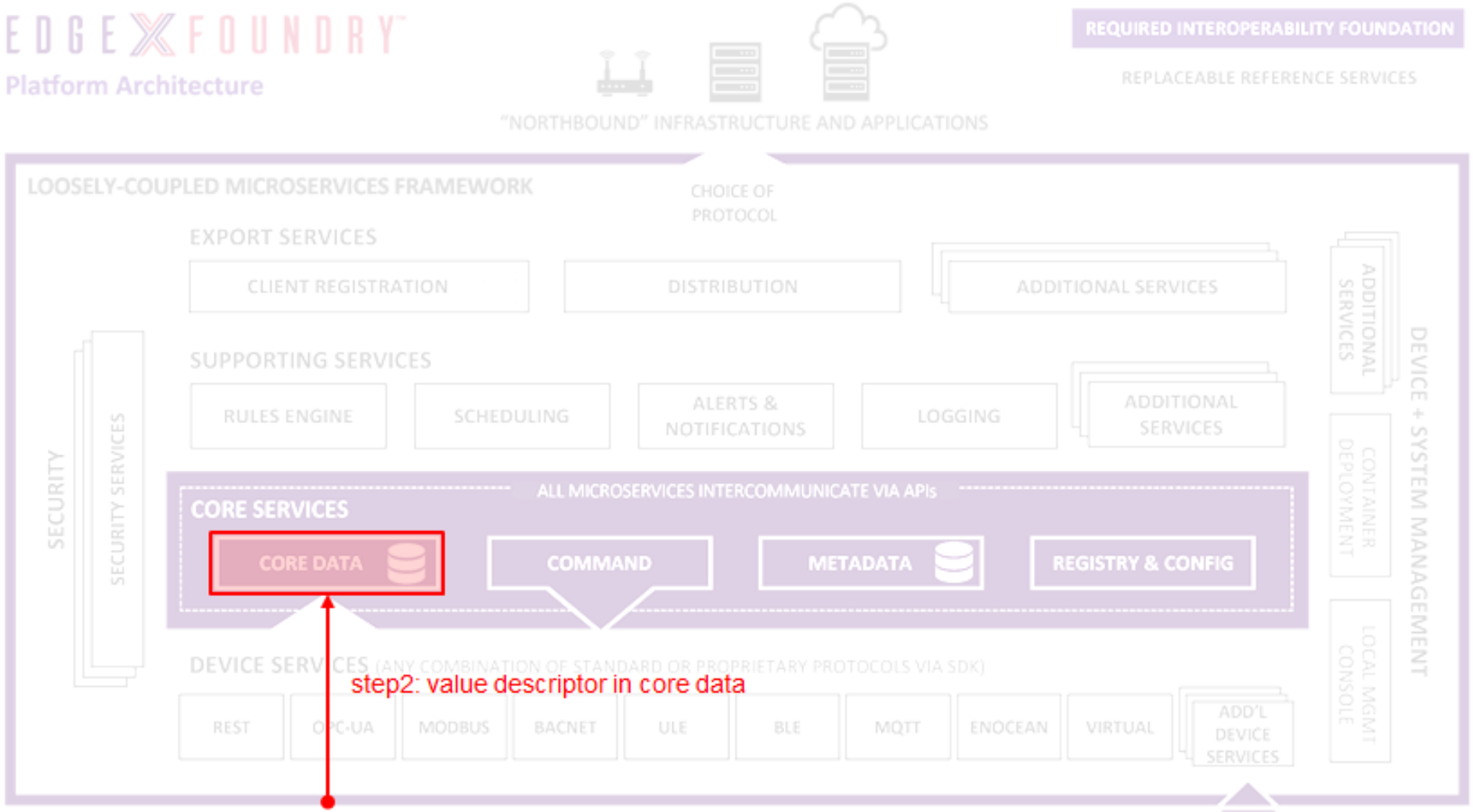
Device and Device Service Setup (2/6)

- **Creating Reference Information in EdgeX**
  - **Addressables**
    - An Addressable is a flexible EdgeX object that specifies a physical address of something
    - Calls **to Core Metadata** to create the Addressable for the Device Service
      - POST to http://localhost:48081/api/v1/addressable
      - BODY: {"name":"camera control", "protocol":"HTTP", "address":"172.17.0.1", "port":49977, "path":"/cameracontrol", "publisher":"none", "user":"none", "password":"none", "topic":"none"}
    - Calls **to Core Metadata** to create the Addressable for the Device (the camera in this case)
      - POST to http://localhost:48081/api/v1/addressable
      - BODY: {"name":"camera1 address", "protocol":"HTTP", "address":"172.17.0.1", "port":49999, "path":"/camera1", "publisher":"none", "user":"none", "password":"none", "topic":"none"}
    - Assumption: both the Device Service and Device are able to be reached via HTTP REST


EDGE X FOUNDRY
Platform Architecture
REQUIRED INTEROPERABILITY FOUNDATION
REPLACEABLE REFERENCE SERVICES
"NORTHBOUND" INFRASTRUCTURE AND APPLICATIONS
LOOSELY-COUPLED MICROSERVICES FRAMEWORK
CHOICE OF PROTOCOL
EXPORT SERVICES
CLIENT REGISTRATION
DISTRIBUTION
ADDITIONAL SERVICES
SUPPORTING SERVICES
RULES ENGINE
SCHEDULING
ALERTS & NOTIFICATIONS
LOGGING
ADDITIONAL SERVICES
SECURITY
SECURITY SERVICES
ADDITIONAL SERVICES
CONTAINER DEPLOYMENT
LOCAL MGMT CONSOLE
DEVICE + SYSTEM MANAGEMENT
ALL MICROSERVICES INTERCOMMUNICATE VIA APIs
CORE SERVICES
CORE DATA
COMMAND
METADATA
REGISTRY & CONFIG
DEVICE SERVICES (ANY COMBINATION OF STANDARD OR PROPRIETARY PROTOCOLS VIA SDK)
MQTT
ENOCEAN
VIRTUAL
ADD'L DEVICE SERVICES
step1: addressable
camera service
49977
POST to http://localhost:48081/api/v1/addressable
BODY: {"name":"camera control","protocol":"HTTP","address":"172.17.0.1","port":49977,"path":"/cameracontrol","publisher":"none","user":"none","password":"none","topic":"none"}
camera 1
49999
POST to http://localhost:48081/api/v1/addressable
BODY: {"name":"camera1 address","protocol":"HTTP","address":"172.17.0.1","port":49999,"path":"/camera1","publisher":"none","user":"none","password":"none","topic":"none"}
SDK
"SOUTHBOUND" DEVICES, SENSORS AND ACTUATORS

Device and Device Service Setup (3/6)

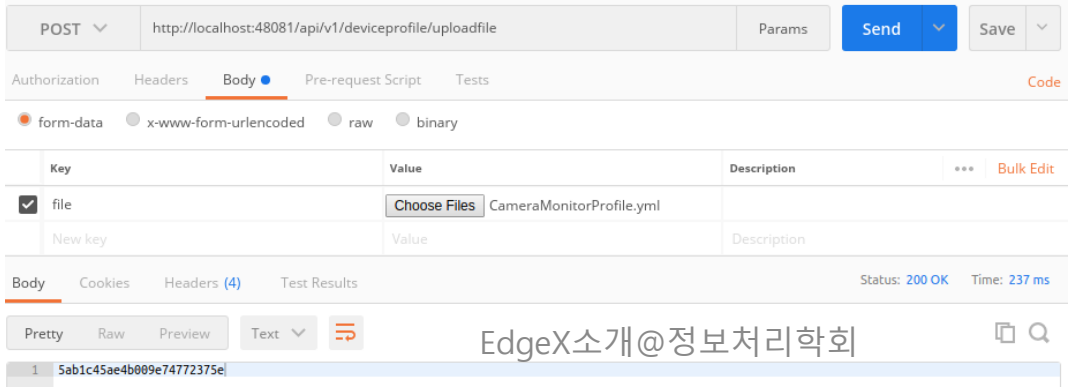
- **Value Descriptors**
  - Provide the context and unit of measure for data (or values) sent to and from a Device
  - Calls **to Core Data** to establish 5 Value Descriptors
    - POST to http://localhost:48080/api/v1/valuedescriptor
    - BODY: {"name":"humancount", "description":"people count", "min":"0", "max":"100", "type":"I", "uomLabel":"count", "defaultValue":"0", "formatting":"%s", "labels":["count","humans"]}
    - POST to http://localhost:48080/api/v1/valuedescriptor
    - BODY: {"name":"caninecount", "description":"dog count", "min":"0", "max":"100", "type":"I", "uomLabel":"count", "defaultValue":"0", "formatting":"%s", "labels":["count","canines"]}
    - POST to http://localhost:48080/api/v1/valuedescriptor
    - BODY: {"name":"depth", "description":"scan distance", "min":"1","max":"10", "type":"I", "uomLabel":"feet", "defaultValue":"1", "formatting":"%s", "labels":["scan","distance"]}
    - POST to http://localhost:48080/api/v1/valuedescriptor
    - BODY: {"name":"duration", "description":"time between events", "min":"10", "max":"180", "type":"I", "uomLabel":"seconds", "defaultValue":"10", "formatting":"%s", "labels":["duration","time"]}
    - POST to http://localhost:48080/api/v1/valuedescriptor
    - BODY: {"name":"cameraerror", "description":"error response message from a camera", "min":"", "max":"", "type":"S", "uomLabel":"", "defaultValue":"error", "formatting":"%s", "labels":["error","message"]}



POST to http://localhost:48080/api/v1/valuedescriptor
BODY: {"name":"humancount","description":"people count", "min":"0","max":"100","type":"I","uomLabel":"count","defaultValue":"0","formatting":"%s","labels":["count","humans"]}
BODY: {"name":"caninecount","description":"dog count", "min":"0","max":"100","type":"I","uomLabel":"count","defaultValue":"0","formatting":"%s","labels":["count","canines"]}
BODY: {"name":"depth","description":"scan distance", "min":"1","max":"10","type":"I","uomLabel":"feet","defaultValue":"1","formatting":"%s","labels":["scan","distance"]}
BODY: {"name":"duration","description":"time between events","min":"10","max":"180","type":"I","uomLabel":"seconds","defaultValue":"10","formatting":"%s","labels":["duration","time"]}
BODY: {"name":"cameraerror","description":"error response message from a camera", "min":"","max":"","type":"S","uomLabel":"","defaultValue":"error","formatting":"%s","labels":["error","message"]}

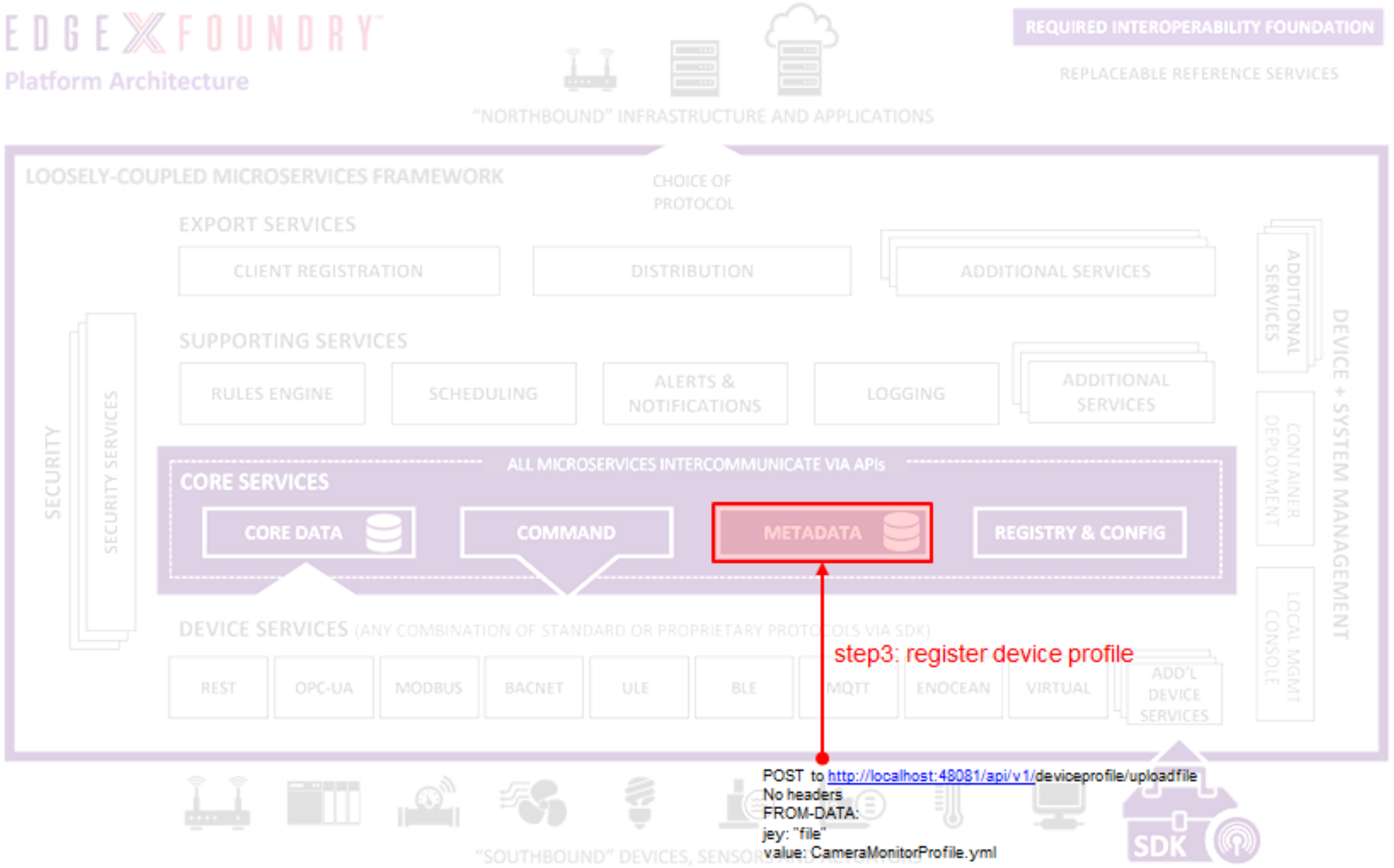
Device and Device Service Setup (4/6)

- **Device Profile**
  - A classification of Device
    - the type of Device, the data theses Devices provide, and how to command them
  - Device Profiles are often represented in YAML
    - POST to http://localhost:48081/api/v1/deviceprofile/uploadfile
    - No headers
    - FORM-DATA:
      - key: file
      - value: CameraMonitorProfile.yml



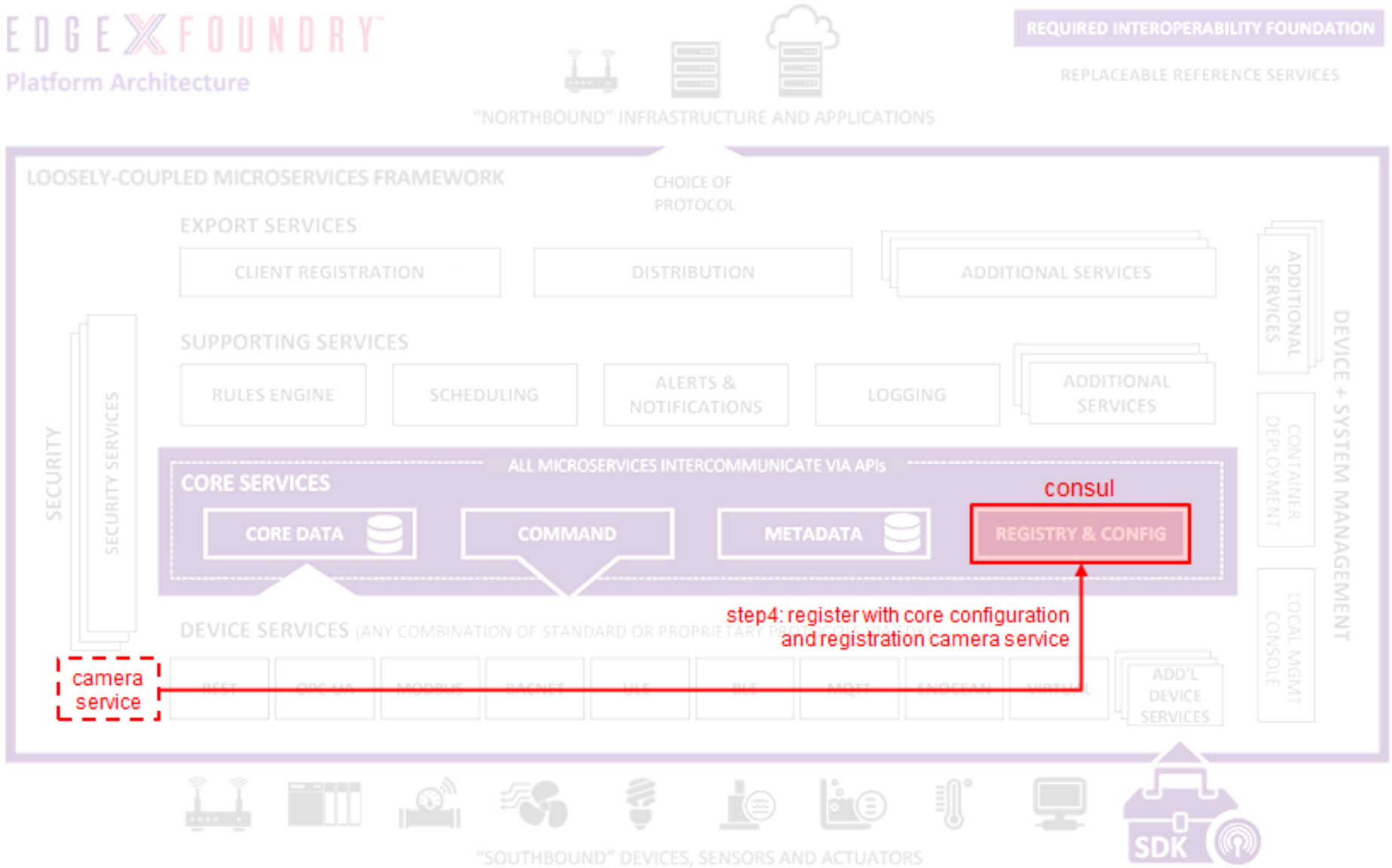
CameraMonitorProfile.yml

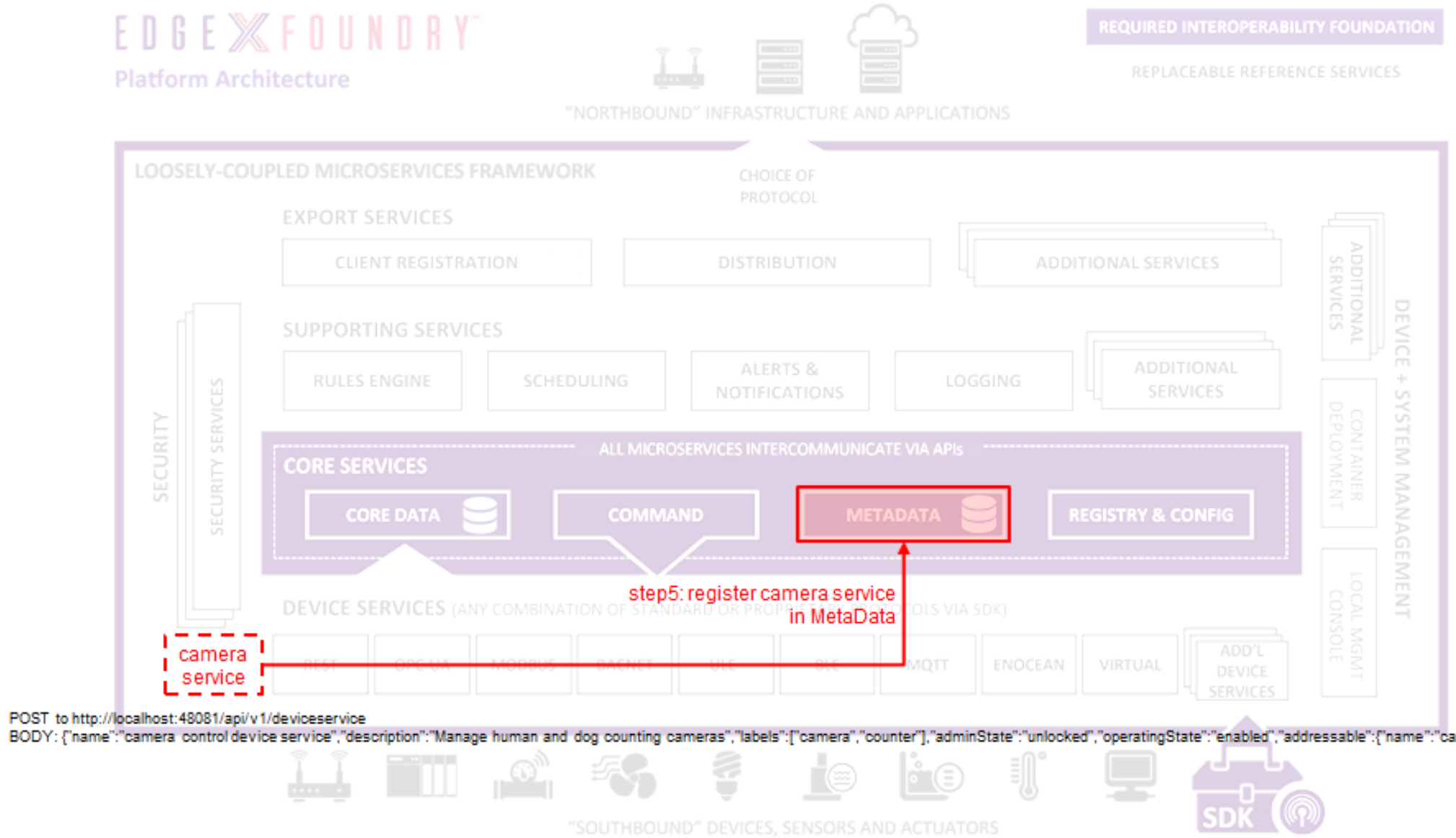
```yaml
# Copyright 2017 Dell Inc. All rights reserved.
name: "camera monitor profile"
manufacturer: "Dell"
model: "Cam12345"
labels:
    - "camera"
description: "Human and canine camera monitor profile"
commands:
  -
    name: People
    get:
        path: "/api/v1/devices/{deviceId}/peoplecount"
        responses:
          -
            code: "200"
            description: "Number of people on camera"
            expectedValues: ["humancount"]
          -
            code: "503"
            description: "service unavailable"
            expectedValues: ["cameraerror"]
  -
    name: Dogs
    get:
        path: "/api/v1/devices/{deviceId}/dogcount"
        responses:
          -
            code: "200"
            description: "Number of dogs on camera"
            expectedValues: ["caninecount"]
          -
            code: "503"
            description: "service unavailable"
            expectedValues: ["cameraerror"]
  -
    name: ScanDepth
    get:
        path: "/api/v1/devices/{deviceId}/scandepth"
        responses:
          -
            code: "200"
            description: "Get the scan depth"
            expectedValues: ["depth"]
          -
            code: "503"
            description: "service unavailable"
            expectedValues: ["cameraerror"]
    put:
        path: "/api/v1/devices/{deviceId}/scandepth"
        parameterNames: ["depth"]
        responses:
          -
            code: "204"
            description: "Set the scan depth."
            expectedValues: []
```


EDGE X FOUNDRY
Platform Architecture
REQUIRED INTEROPERABILITY FOUNDATION
REPLACEABLE REFERENCE SERVICES
"NORTHBOUND" INFRASTRUCTURE AND APPLICATIONS
LOOSELY-COUPLED MICROSERVICES FRAMEWORK
CHOICE OF PROTOCOL
EXPORT SERVICES
CLIENT REGISTRATION
DISTRIBUTION
ADDITIONAL SERVICES
SUPPORTING SERVICES
RULES ENGINE
SCHEDULING
ALERTS & NOTIFICATIONS
LOGGING
ADDITIONAL SERVICES
SECURITY
SECURITY SERVICES
DEVICE + SYSTEM MANAGEMENT
ADDITIONAL SERVICES
CONTAINER DEPLOYMENT
LOCAL MGMT CONSOLE
ALL MICROSERVICES INTERCOMMUNICATE VIA APIs
CORE SERVICES
CORE DATA
COMMAND
METADATA
REGISTRY & CONFIG
DEVICE SERVICES (ANY COMBINATION OF STANDARD OR PROPRIETARY PROTOCOLS VIA SDK)
REST
OPC-UA
MODBUS
BACNET
ULE
BLE
MQTT
ENOCEAN
VIRTUAL
ADD'L DEVICE SERVICES
step3: register device profile
POST to http://localhost:48081/api/v1/deviceprofile/uploadfile
No headers
FROM-DATA:
jey: "file"
value: CameraMonitorProfile.yml
SDK
"SOUTHBOUND" DEVICES, SENSORS AND ACTUATORS

Device and Device Service Setup (5/6)

- **Create the Device Service in EdgeX**
  - Once the reference information is established, the Device Service can register or define itself in EdgeX
  - **Register with Core Configuration and Registration Micro Service (Core Configuration and Registry API)**
    - Provides its location to the Config/Reg micro service and picks up any new/latest configuration information from this central service.
    - Since there is no real Device Service in this demonstration, this part of the inter-micro service exchange is not explored here
  - **Create the Device Service in Metadata**
    - The Device Service is associated to the Addressable for the Device Service that is already Core Metadata.
    - POST to http://localhost:48081/api/v1/deviceservice
    - BODY: {"name":"camera control device service", "description":"Manage human and dog counting cameras", "labels":["camera","counter"], "adminState":"unlocked", "operatingState":"enabled", "addressable":{"name":"camera control"}}

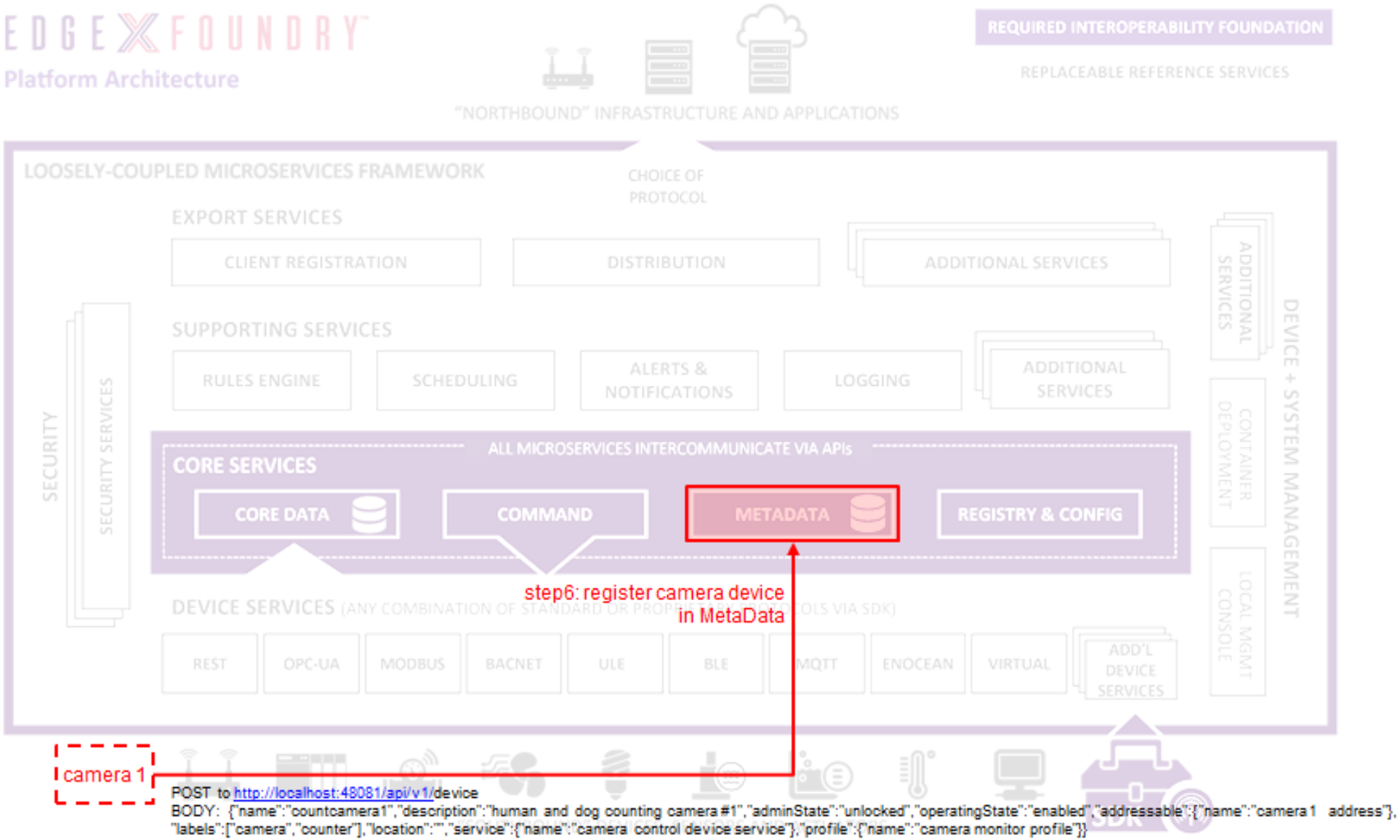

EDGE X FOUNDRY™
Platform Architecture
REQUIRED INTEROPERABILITY FOUNDATION
REPLACEABLE REFERENCE SERVICES
"NORTHBOUND" INFRASTRUCTURE AND APPLICATIONS
LOOSELY-COUPLED MICROSERVICES FRAMEWORK
CHOICE OF PROTOCOL
EXPORT SERVICES
CLIENT REGISTRATION
DISTRIBUTION
ADDITIONAL SERVICES
SUPPORTING SERVICES
RULES ENGINE
SCHEDULING
ALERTS & NOTIFICATIONS
LOGGING
ADDITIONAL SERVICES
SECURITY
SECURITY SERVICES
ALL MICROSERVICES INTERCOMMUNICATE VIA APIs
CORE SERVICES
consul
CORE DATA
COMMAND
METADATA
REGISTRY & CONFIG
step4: register with core configuration and registration camera service
DEVICE SERVICES (ANY COMBINATION OF STANDARD OR PROPRIETARY PROTOCOLS)
camera service
REST
OPC-UA
MODBUS
BACNET
IOT
BLE
MQTT
ENOCEAN
VIRTUAL
ADD'L DEVICE SERVICES
ADDITIONAL SERVICES
CONTAINER DEPLOYMENT
LOCAL MGMT CONSOLE
DEVICE + SYSTEM MANAGEMENT
SDK
"SOUTHBOUND" DEVICES, SENSORS AND ACTUATORS



POST to http://localhost:48081/api/v1/deviceservice
BODY: {"name":"camera control device service","description":"Manage human and dog counting cameras","labels":["camera","counter"],"adminState":"unlocked","operatingState":"enabled","addressable":{"name":"camera control"}}

Device and Device Service Setup (6/6)

- **Provision a Device**
  - Lastly, a Device Service *often* discovers and provisions new Devices it finds
  - To create a Device, it must be associated to
    - a Device Profile (by name or id),
    - a Device Service (by name or id),
    - and Addressable (by name or id).
    - POST to http://localhost:48081/api/v1/device
    - BODY: {"name":"countcamera1", "description":"human and dog counting camera #1", "adminState":"unlocked", "operatingState":"enabled", "addressable":{"name":"camera1 address"}, "labels":["camera","counter"], "location":"", "service":{"name":"camera control device service"}, "profile":{"name":"camera monitor profile"}}


EDGE X FOUNDRY
Platform Architecture
REQUIRED INTEROPERABILITY FOUNDATION
REPLACEABLE REFERENCE SERVICES
"NORTHBOUND" INFRASTRUCTURE AND APPLICATIONS
LOOSELY-COUPLED MICROSERVICES FRAMEWORK
CHOICE OF PROTOCOL
EXPORT SERVICES
CLIENT REGISTRATION
DISTRIBUTION
ADDITIONAL SERVICES
SUPPORTING SERVICES
RULES ENGINE
SCHEDULING
ALERTS & NOTIFICATIONS
LOGGING
ADDITIONAL SERVICES
SECURITY
SECURITY SERVICES
ALL MICROSERVICES INTERCOMMUNICATE VIA APIs
CORE SERVICES
CORE DATA
COMMAND
METADATA
REGISTRY & CONFIG
DEVICE SERVICES
REST
OPC-UA
MODBUS
BACNET
ULE
BLE
MQTT
ENOCEAN
VIRTUAL
ADD'L DEVICE SERVICES
DEVICE + SYSTEM MANAGEMENT
CONTAINER DEPLOYMENT
LOCAL MGMT CONSOLE
step6: register camera device in MetaData
camera 1
POST to http://localhost:48081/api/v1/device
BODY: {"name":"countcamera1","description":"human and dog counting camera #1","adminState":"unlocked","operatingState":"enabled","addressable":{"name":"camera 1 address"},
"labels":["camera","counter"],"location":"","service":{"name":"camera control device service"},"profile":{"name":"camera monitor profile"}}

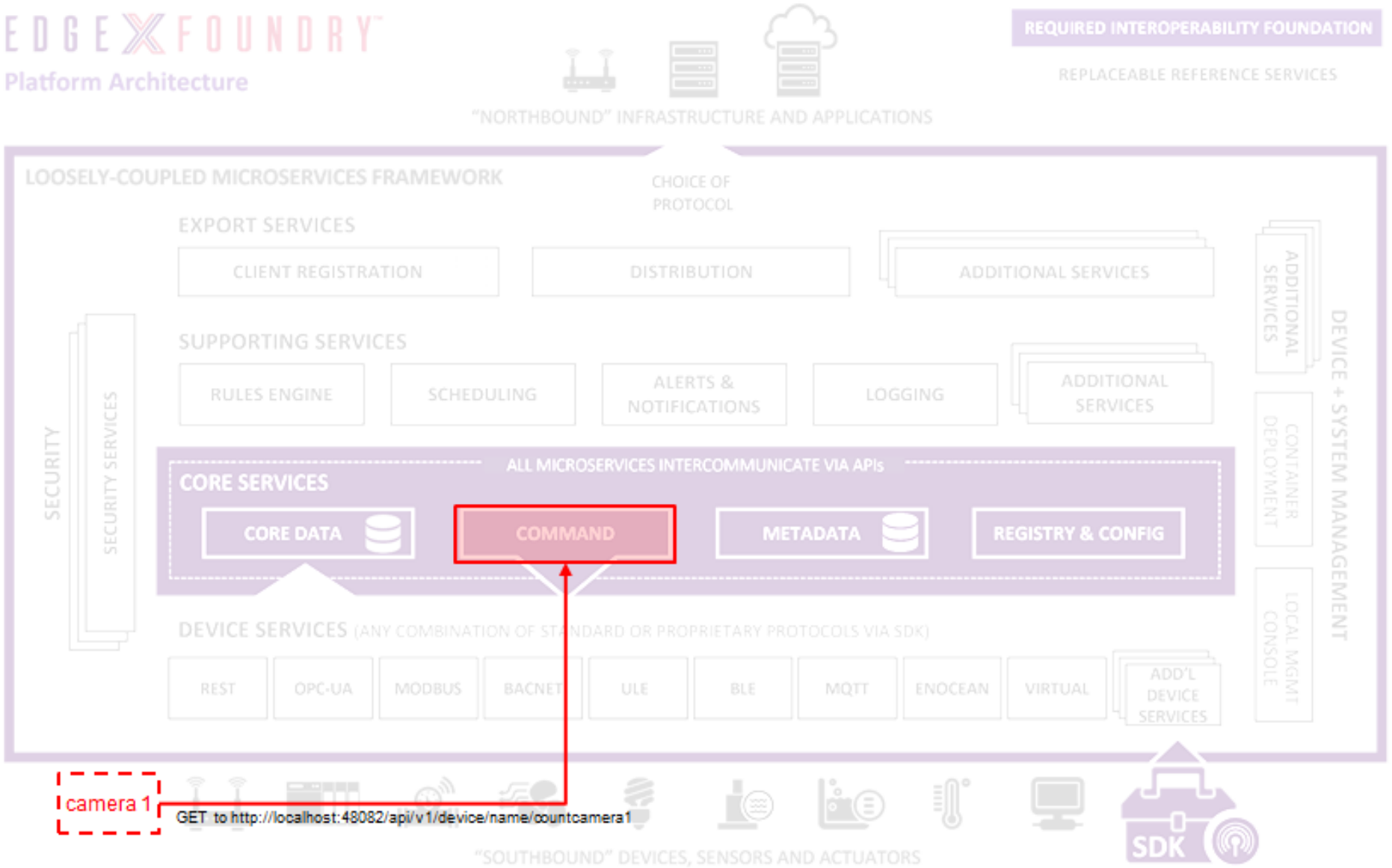
Test the Setup

- Let's try a few of the micro service APIs out to confirm that things have been configured correctly
  - **Check the Device Service**
    - GET to http://localhost:48081/api/v1/deviceservice
    - GET to http://localhost:48081/api/v1/deviceservice/label/camera
  - **Check the Device**
    - GET to http://localhost:48081/api/v1/device
    - GET to http://localhost:48081/api/v1/device/profilename/camera monitor profile
  - **Check the Commands**
    - GET to http://localhost:48082/api/v1/device/name/countcamera1
  - **Check the Value Descriptors**
    - GET to http://localhost:48080/api/v1/valuedescriptor
    - GET to http://localhost:48080/api/v1/event/count
      - Returns 0 (no data yet)

Execute a Command (1/2)

- Let's launch a Command to set the scan depth of countcamera1
  - First, get the URL for the Command to set a new scan depth on the Device
    - GET to http://localhost:48082/api/v1/device/name/countcamera1

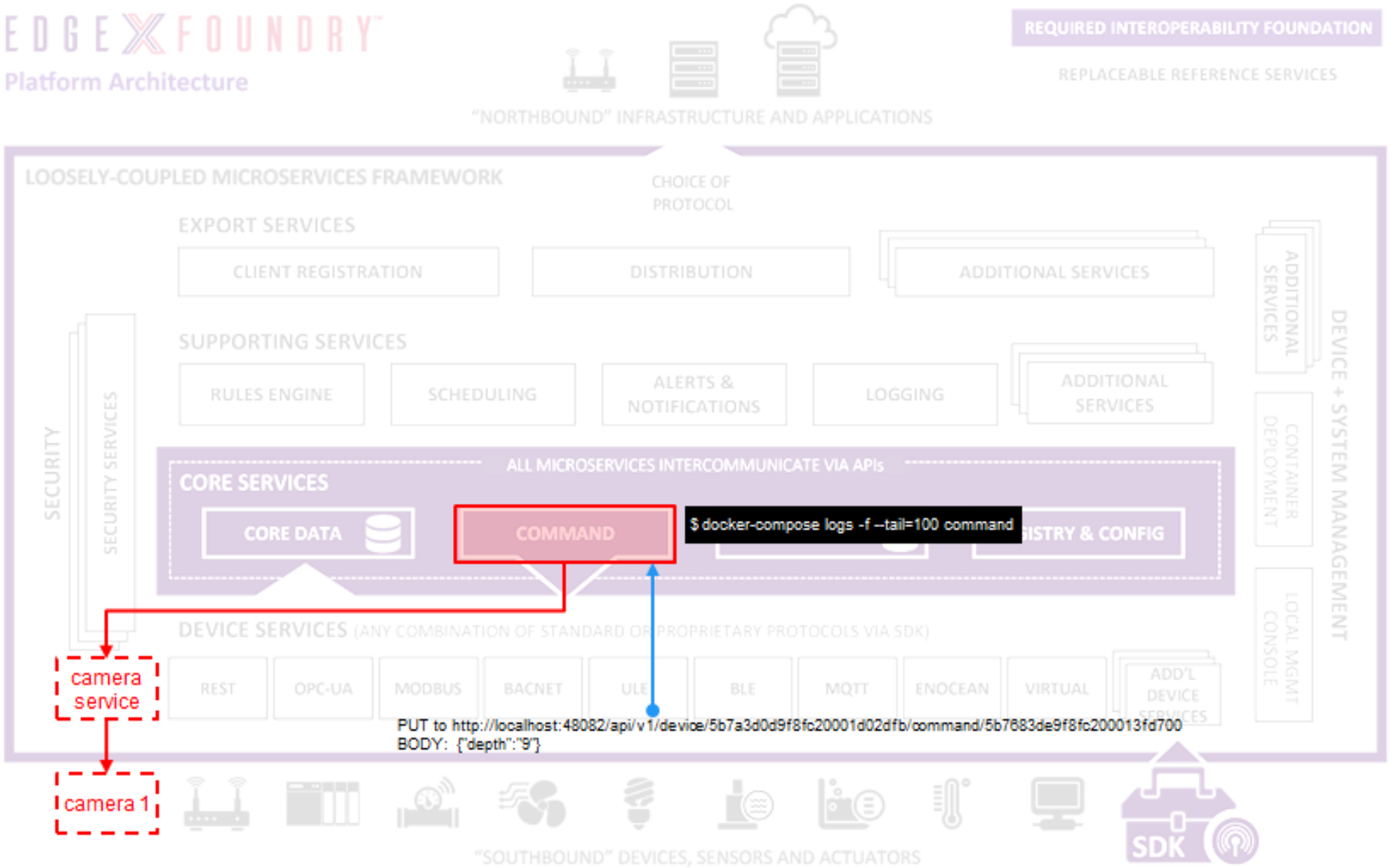

```
                    "expectedValues": [
                        "cameraerror"
                    ]
                }
            ]
        },
        "put": {
            "url": "http://192.168.99.100:48082/api/v1/device/59625992e4b0c3937c3ac446/command/596258f1e4b0c3937c3ac441",
            "parameterNames": [
                "depth"
            ],
            "responses": [
                {
                    "code": "204",
                    "description": "Set the scan depth.",
                    "expectedValues": []
                },
                {
                    "code": "503",
                    "description": "service unavailable",
                    "expectedValues": [
                        "cameraerror"
                    ]
```


EDGE X FOUNDRY™
Platform Architecture
REQUIRED INTEROPERABILITY FOUNDATION
REPLACEABLE REFERENCE SERVICES
"NORTHBOUND" INFRASTRUCTURE AND APPLICATIONS
LOOSELY-COUPLED MICROSERVICES FRAMEWORK
CHOICE OF PROTOCOL
EXPORT SERVICES
CLIENT REGISTRATION
DISTRIBUTION
ADDITIONAL SERVICES
SUPPORTING SERVICES
RULES ENGINE
SCHEDULING
ALERTS & NOTIFICATIONS
LOGGING
ADDITIONAL SERVICES
ADDITIONAL SERVICES
CONTAINER DEPLOYMENT
LOCAL MGMT CONSOLE
DEVICE + SYSTEM MANAGEMENT
SECURITY
SECURITY SERVICES
ALL MICROSERVICES INTERCOMMUNICATE VIA APIs
CORE SERVICES
CORE DATA
COMMAND
METADATA
REGISTRY & CONFIG
DEVICE SERVICES (ANY COMBINATION OF STANDARD OR PROPRIETARY PROTOCOLS VIA SDK)
REST
OPC-UA
MODBUS
BACNET
ULE
BLE
MQTT
ENOCEAN
VIRTUAL
ADD'L DEVICE SERVICES
camera 1
GET to http://localhost:48082/api/v1/device/name/countcamera1
SDK
"SOUTHBOUND" DEVICES, SENSORS AND ACTUATORS

Execute a Command (2/2)

- Copy this URL into your REST client tool of choice and make a PUT
  - PUT to http://localhost:48082/api/v1/device/<system specific device id>/command/<system specific command id>
  - BODY: {"depth":"9"}
  - Because no Device Service (or Device) actually exists, Core Command will respond with an HTTP 503 Service Unavailable error (Service issue: Connection refused)
  - Use Support Logging service to prove that
    - the Core Command micro service did receive the request and attempted to issue the actuating command
    - GET to http://localhost:48061/api/v1/logs/logLevels/INFO/originServices/edgex-core-command/0/9000000000000/100


```
{
    "id": null,
    "logLevel": "INFO",
    "labels": null,
    "originService": "edgex-core-command",
    "message": "Issuing put command to: HTTP://172.17.0.1:49977/api/v1/devices/59625992e4b0c3937c3ac446/scandepth",
    "created": 1499621634876
},
{
    "id": null,
    "logLevel": "INFO",
    "labels": null,
    "originService": "edgex-core-command",
    "message": "Command message body is:  {\"depth\":\"9\"}",
    "created": 1499621634926
},
```


EDGE X FOUNDRY™
Platform Architecture
REQUIRED INTEROPERABILITY FOUNDATION
REPLACEABLE REFERENCE SERVICES
"NORTHBOUND" INFRASTRUCTURE AND APPLICATIONS
LOOSELY-COUPLED MICROSERVICES FRAMEWORK
CHOICE OF PROTOCOL
EXPORT SERVICES
CLIENT REGISTRATION
DISTRIBUTION
ADDITIONAL SERVICES
SUPPORTING SERVICES
RULES ENGINE
SCHEDULING
ALERTS & NOTIFICATIONS
LOGGING
ADDITIONAL SERVICES
SECURITY
SECURITY SERVICES
ADDITIONAL SERVICES
CONTAINER DEPLOYMENT
LOCAL MGMT CONSOLE
DEVICE + SYSTEM MANAGEMENT
ALL MICROSERVICES INTERCOMMUNICATE VIA APIs
CORE SERVICES
CORE DATA
COMMAND
$ docker-compose logs -f --tail=100 command
ISTRY & CONFIG
DEVICE SERVICES (ANY COMBINATION OF STANDARD OR PROPRIETARY PROTOCOLS VIA SDK)
REST
OPC-UA
MODBUS
BACNET
ULE
BLE
MQTT
ENOCEAN
VIRTUAL
ADD'L DEVICE SERVICES
camera service
camera 1
PUT to http://localhost:48082/api/v1/device/5b7a3d0d9f8fc20001d02dfb/command/5b7683de9f8fc200013fd700
BODY: {"depth":"9"}
SDK
"SOUTHBOUND" DEVICES, SENSORS AND ACTUATORS

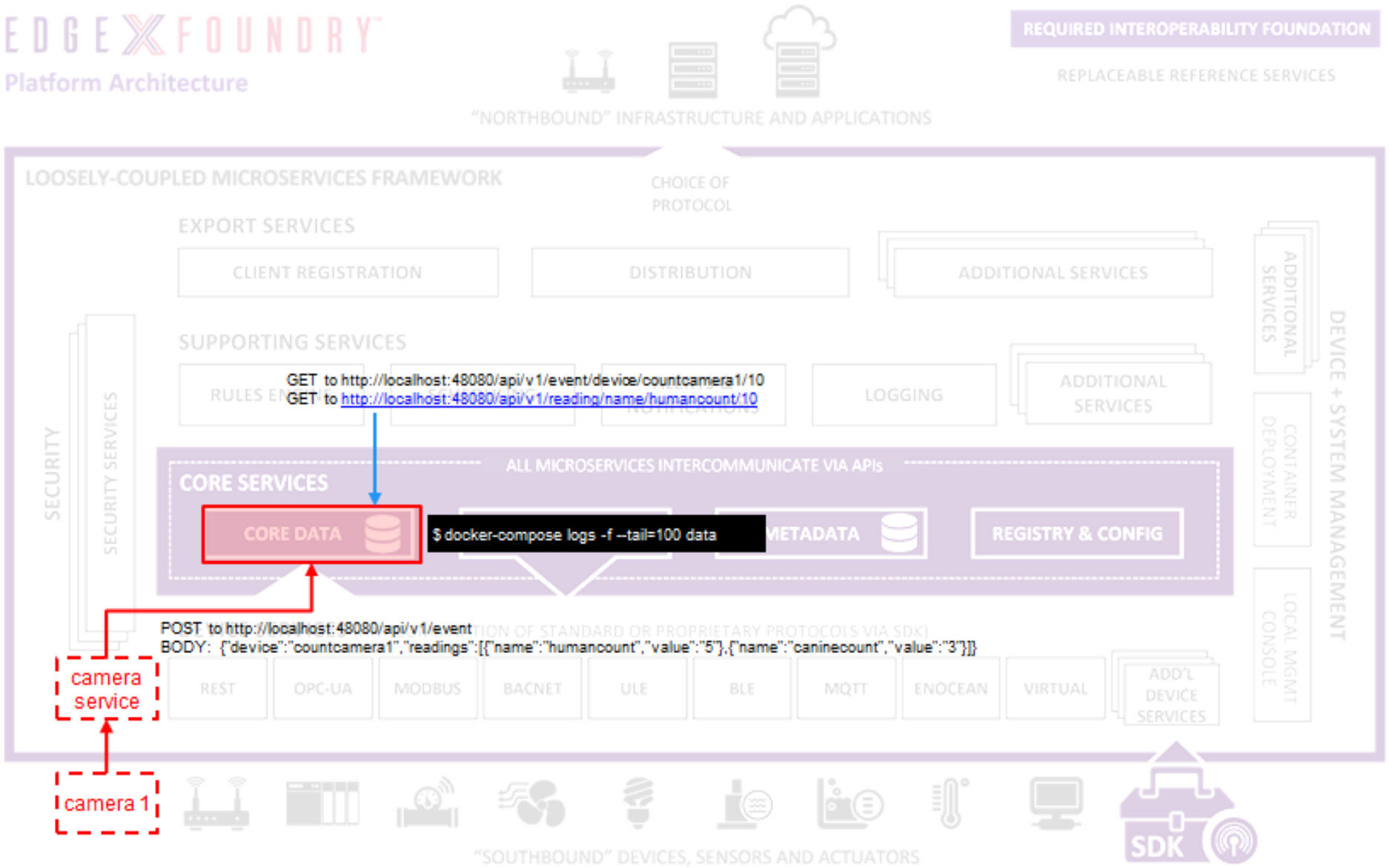
Send an Event/Reading

- Data is submitted **to Core Data** as an Event
  - Event: a collection of sensor readings from a Device (associated to a Device by its ID or name) at a particular point in time
  - Reading in an Event: a particular value sensed by the Device and associated to a Value Descriptor (by name)
- The Device Service, on creating the Event and associated Reading objects would transmit this information to Core Data via REST call
  - POST to http://localhost:48080/api/v1/event
  - BODY: {"device":"countcamera1", "readings":[{"name":"humancount","value":"5"}, {"name":"caninecount","value":"3"}]}
- The Device Service can also supply an origin property
  - Time (in Epoch timestamp/milliseconds format) at which the data was sensed/collected
  - BODY: {"device":"countcamera1", "origin":1471806386919, "readings":[{"name":"humancount", "value":"1","origin":1471806386919}, {"name":"caninecount","value":"0", "origin":1471806386919}]}

Explore Core Data

- You can use the **Core Data** API to explore that data that is now stored in MongoDB

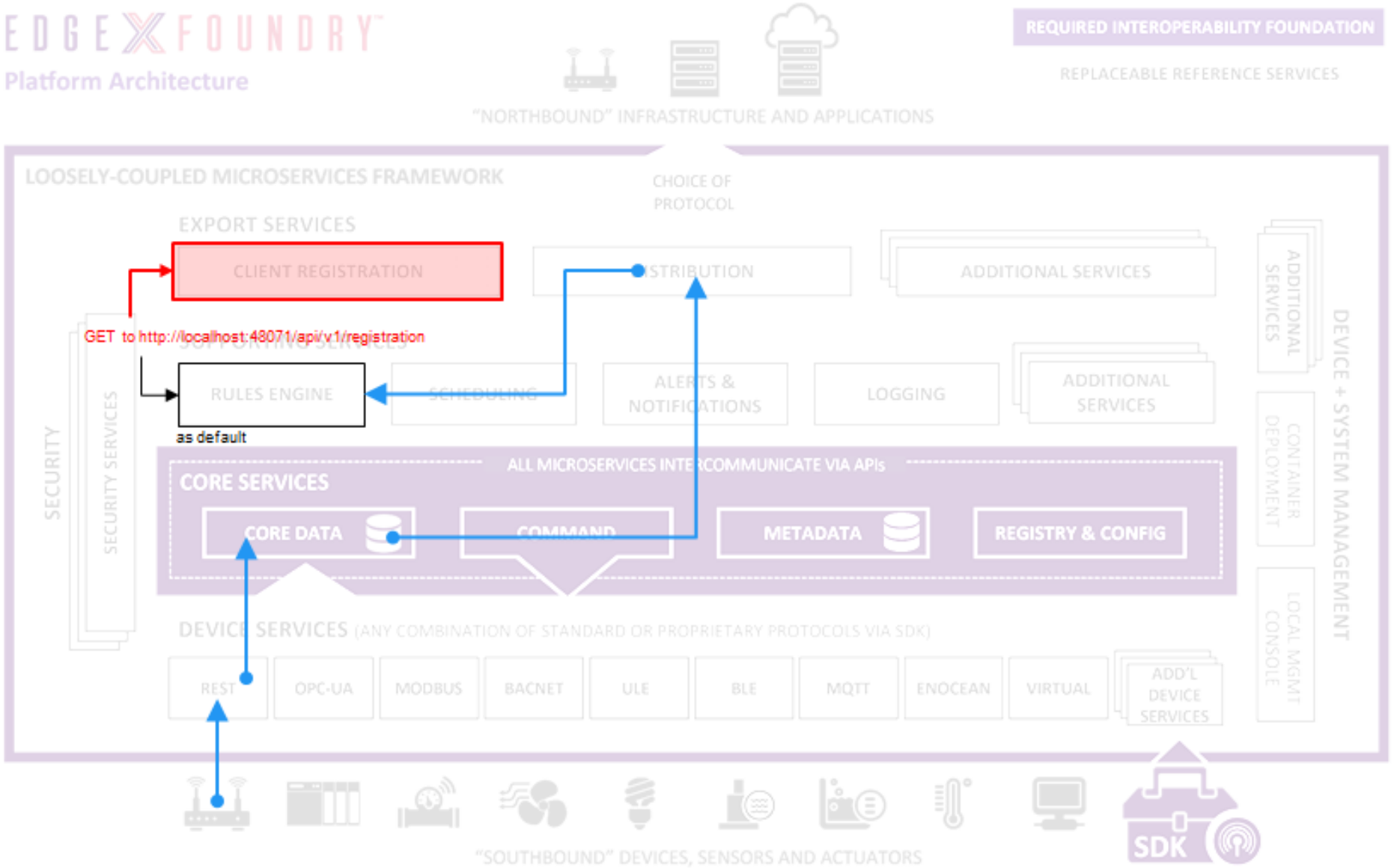
- Two Event records should be the count returned
  - GET to http://localhost:48080/api/v1/event/count
- Retrieve 10 of the Events associated to the countcamera1 Device.
  - GET to http://localhost:48080/api/v1/event/device/countcamera1/10
- Retrieve 10 of the human count Readings associated to the countcamera1 Device (i.e. - get Readings by Value Descriptor)
  - Get to http://localhost:48080/api/v1/reading/name/humancount/10


EDGE X FOUNDRY™
Platform Architecture
REQUIRED INTEROPERABILITY FOUNDATION
REPLACEABLE REFERENCE SERVICES
"NORTHBOUND" INFRASTRUCTURE AND APPLICATIONS
LOOSELY-COUPLED MICROSERVICES FRAMEWORK
CHOICE OF PROTOCOL
EXPORT SERVICES
CLIENT REGISTRATION
DISTRIBUTION
ADDITIONAL SERVICES
SUPPORTING SERVICES
RULES ENGINE
NOTIFICATIONS
LOGGING
ADDITIONAL SERVICES
ADDITIONAL SERVICES
CONTAINER DEPLOYMENT
LOCAL MGMT CONSOLE
DEVICE + SYSTEM MANAGEMENT
SECURITY
SECURITY SERVICES
GET to http://localhost:48080/api/v1/event/device/countcamera1/10
GET to http://localhost:48080/api/v1/reading/name/humancount/10
ALL MICROSERVICES INTERCOMMUNICATE VIA APIs
CORE SERVICES
CORE DATA
$ docker-compose logs -f --tail=100 data
METADATA
REGISTRY & CONFIG
POST to http://localhost:48080/api/v1/event
BODY: {"device":"countcamera1","readings":[{"name":"humancount","value":"5"},{"name":"caninecount","value":"3"}]}
camera service
camera 1
REST
OPC-UA
MODBUS
BACNET
ULE
BLE
MQTT
ENOCEAN
VIRTUAL
ADD'L DEVICE SERVICES
SDK
"SOUTHBOUND" DEVICES, SENSORS AND ACTUATORS

Register an Export Client (1/2)

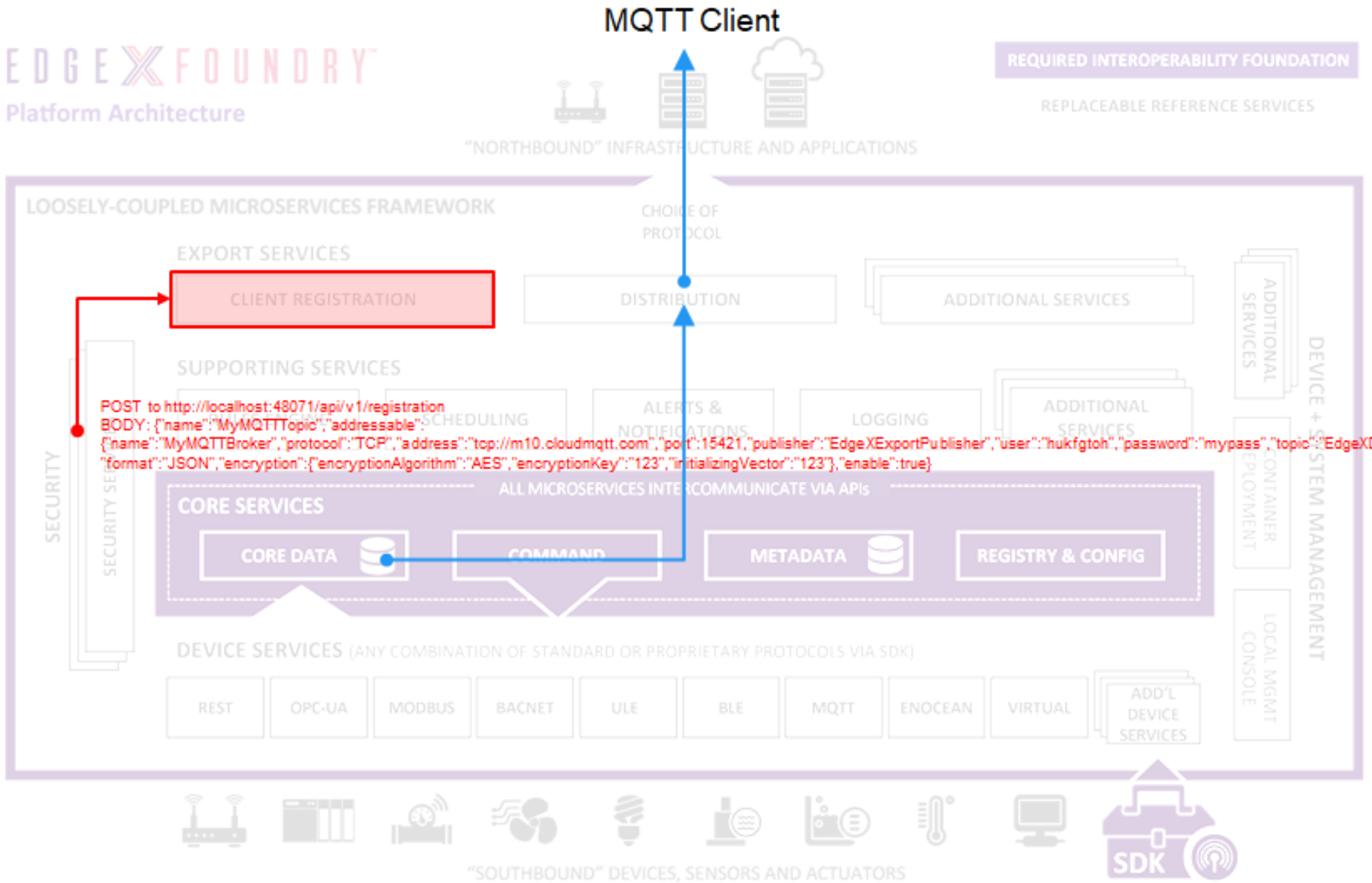
- *Anything* wishing to receive the sensor/device data as it comes into EdgeX must register as an "export" client
  - An enterprise system or the Cloud
  - A edge analytics system (like the Rules Engine) that actuate on a Device
- A special client - Rules Engine
  - Automatically registered as a client of the export services
  - Receives all the Events/Readings from Core Data sent by Devices
- Request a list from the Export Client micro service
  - GET to http://localhost:48071/api/v1/registration


```
[
    {
        "id": "59627a4fe4b01d977d80ba06",
        "created": 1499626063114,
        "modified": 1499626063114,
        "origin": 0,
        "name": "EdgeXRulesEngine",
        "addressable": {
            "id": null,
            "created": 0,
            "modified": 0,
            "origin": 0,
            "name": "EdgeXRulesEngineAddressable",
            "method": "POST",
            "protocol": "ZMQ",
            "address": "",
            "port": 0,
            "path": "",
            "publisher": null,
            "user": null,
            "password": null,
            "topic": null,
            "baseURL": "ZMQ://:0",
            "url": "ZMQ://:0"
        },
        "format": "SERIALIZED",
        "filter": null,
        "encryption": null,
        "compression": "NONE",
        "enable": true,
        "destination": "ZMQ_TOPIC"
    }
]
```


EDGE X FOUNDRY™
Platform Architecture
REQUIRED INTEROPERABILITY FOUNDATION
REPLACEABLE REFERENCE SERVICES
"NORTHBOUND" INFRASTRUCTURE AND APPLICATIONS
LOOSELY-COUPLED MICROSERVICES FRAMEWORK
CHOICE OF PROTOCOL
EXPORT SERVICES
CLIENT REGISTRATION
DISTRIBUTION
ADDITIONAL SERVICES
GET to http://localhost:48071/api/v1/registration
RULES ENGINE
as default
SCHEDULING
ALERTS & NOTIFICATIONS
LOGGING
ADDITIONAL SERVICES
ADDITIONAL SERVICES
CONTAINER DEPLOYMENT
LOCAL MGMT CONSOLE
DEVICE + SYSTEM MANAGEMENT
SECURITY
SECURITY SERVICES
ALL MICROSERVICES INTERCOMMUNICATE VIA APIs
CORE SERVICES
CORE DATA
COMMAND
METADATA
REGISTRY & CONFIG
DEVICE SERVICES (ANY COMBINATION OF STANDARD OR PROPRIETARY PROTOCOLS VIA SDK)
REST
OPC-UA
MODBUS
BACNET
ULE
BLE
MQTT
ENOCEAN
VIRTUAL
ADD'L DEVICE SERVICES
SDK
"SOUTHBOUND" DEVICES, SENSORS AND ACTUATORS

Register an Export Client (2/2)

- Register a new client
  - First need to setup
    - a client capable of receiving HTTP REST calls,
    - or an MQTT topic capable of receiving messages from EdgeX
  - MQTT example
    - Assume that an cloud based MQTT Topic has been setup to receive EdgeX Event/Reading data
    - Request Export Client to make a new EdgeX client
      - POST to http://localhost:48071/api/v1/registration
      - BODY: {"name":"MyMQTTTopic", "addressable":{"name":"MyMQTTBroker", "protocol":"TCP", "address":"tcp://m10.cloudmqtt.com", "port":15421, "publisher":"EdgeXExportPublisher", "user":"hukfgtoh", "password":"mypass", "topic":"EdgeXDataTopic"}, "format":"JSON", "encryption":{"encryptionAlgorithm":"AES", "encryptionKey":"123", "initializingVector":"123"}, "enable":true}
    - You can check the Export Distro log to see the attempt was made
      - GET to http://localhost:48061/api/v1/logs/logLevels/INFO/originServices/edgex-export-distro/0/9000000000000/100
      - MQTTOutboundServiceActivator: message sent to MQTT broker: Addressable [name=MyMQTTBroker, protocol=TCP, address=tcp://m10.cloudmqtt.com, port=15421, path=null, publisher=EdgeXExportPublisher, user=hukfgtoh, password=mypass, topic=EdgeXDataTopic, toString()=BaseObject [id=null, created=0, modified=0, origin=0]] : 596283c7e4b0011866276e9


MQTT Client
EDGE X FOUNDRY
Platform Architecture
REQUIRED INTEROPERABILITY FOUNDATION
REPLACEABLE REFERENCE SERVICES
"NORTHBOUND" INFRASTRUCTURE AND APPLICATIONS
LOOSELY-COUPLED MICROSERVICES FRAMEWORK
CHOICE OF PROTOCOL
EXPORT SERVICES
CLIENT REGISTRATION
DISTRIBUTION
ADDITIONAL SERVICES
SUPPORTING SERVICES
POST to http://localhost:48071/api/v1/registration
BODY: {"name":"MyMQTTTopic","addressable":
{"name":"MyMQTTBroker","protocol":"TCP","address":"tcp://m10.cloudmqtt.com","port":15421,"publisher":"EdgeXExportPublisher","user":"hukfgtoh","password":"mypass","topic":"EdgeXDataTopic"},
"format":"JSON","encryption":{"encryptionAlgorithm":"AES","encryptionKey":"123","initializingVector":"123"},"enable":true}
SCHEDULING
ALERTS & NOTIFICATIONS
LOGGING
ADDITIONAL SERVICES
ALL MICROSERVICES INTERCOMMUNICATE VIA APIs
CORE SERVICES
CORE DATA
COMMAND
METADATA
REGISTRY & CONFIG
SECURITY
SECURITY SERVICE
DEVICE + SYSTEM MANAGEMENT
CONTAINER DEPLOYMENT
LOCAL MGMT CONSOLE
DEVICE SERVICES (ANY COMBINATION OF STANDARD OR PROPRIETARY PROTOCOLS VIA SDK)
REST
OPC-UA
MODBUS
BACNET
ULE
BLE
MQTT
ENOCEAN
VIRTUAL
ADD'L DEVICE SERVICES
SDK
"SOUTHBOUND" DEVICES, SENSORS AND ACTUATORS

주요 프로젝트 링크

Access the code:

https://github.com/edgexfoundry

Access the technical documentation:

https://docs.edgexfoundry.org/

Access technical video tutorials:

https://wiki.edgexfoundry.org/display/FA/EdgeX+Tech+Talks

EdgeX Blog:

https://www.edgexfoundry.org/news/blog/

Join an email distribution:

https://lists.edgexfoundry.org/mailman/listinfo

Join the Rocket Chat:

https://chat.edgexfoundry.org/home

Become a project member:

https://www.edgexfoundry.org/about/members/join/

LinkedIn:

https://www.linkedin.com/company/edgexfoundry/

Twitter:

https://twitter.com/EdgeXFoundry

Youtube:

https://www.youtube.com/edgexfoundry

감사합니다.

Appendix

EdgeX 설치 및 실행
– 사용자 버전 –

(참고) EdgeX에 관한 자료는 EdgeX Foundry (https://edgexfoundry.org) 자료를 인용하였습니다.

Docker compose로 설치

#1 – Install Docker & Docker Compose

#2 – Download the EdgeX Foundry Compose File

#3 – Run EdgeX using Compose commands

What You Need
Compose file location: https://github.com/edgexfoundry/developer-scripts/blob/master/compose-files/docker-compose.yml
Memory: minimum of 4 GB
Hard drive space: minimum of 3 GB of space to run the EdgeX Foundry containers, but you may want more depending on how long sensor and device data is retained
OS: EdgeX Foundry has been run successfully on many systems including, but not limited to the following systems - Windows (ver 7 - 10) - Ubuntu Desktop (ver 14-16) - Ubuntu Server (ver 14) - Ubuntu Core (ver 16) - Mac OS X 10

EdgeX 실행하기 (1/2)

Docker command	Description
docker-compose pull	Pull down, but don't start, all the EdgeX Foundry microservices

docker-compose up -d volume	Start the EdgeX Foundry file volume--must be done before the other services are started
	`Jim@JimsXPS MINGW64 ~/edgexfoundry/developer-scripts (master)`
	`$ docker-compose up -d volume`
	`Creating network "developerscripts_edgex-network" with driver "bridge"`
	`Creating edgex-files`

docker-compose up -d config-seed	Start and populate the configuration/registry microservice which all services must register with and get their configuration from
docker-compose up -d mongo	Start the NoSQL MongoDB container
docker-compose up -d logging	Start the logging microservice--used by all micro services that make log entries
docker-compose up -d notifications	Start the notifications and alerts microservice--used by many of the microservices
docker-compose up -d metadata	Start the Core Metadata microservices.
docker-compose up -d data	Start the Core Data microservice

EdgeX 실행하기 (2/2)

Docker command	Description
docker-compose up -d command	Start the Core Command microservice
docker-compose up -d scheduler	Start the scheduling microservice--used by many of the microservices
docker-compose up -d export-client	Start the Export Client registration microservice.
docker-compose up -d export-distro	Start the Export Distribution microservice.
docker-compose up -d rulesengine	Start the Rules Engine microservice.
docker-compose up -d device-virtual	Start the virtual device service

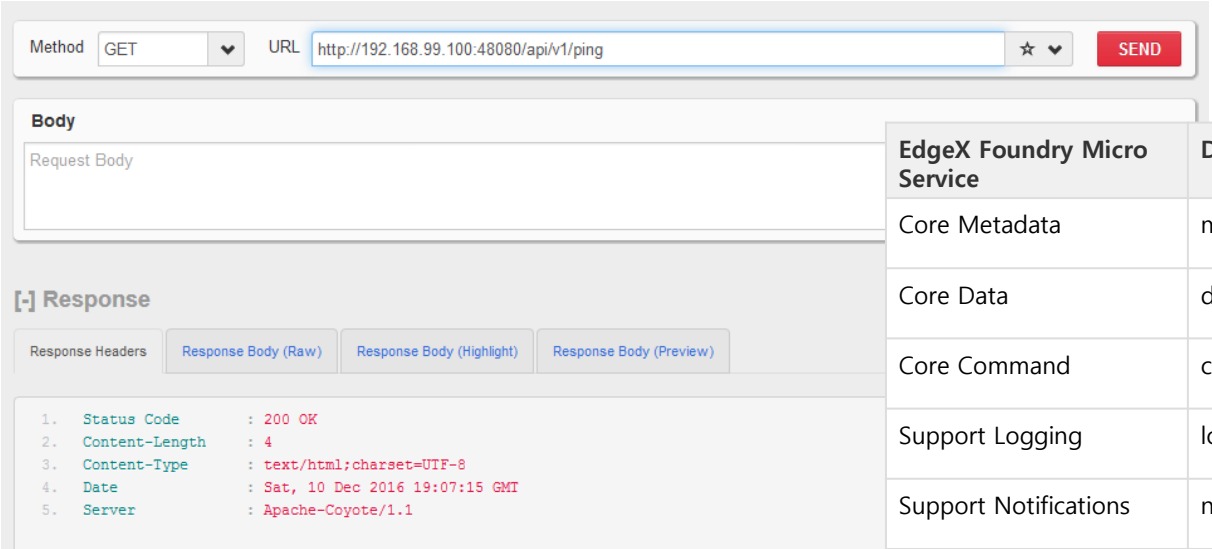
상태 점검 (1/3)

- EdgeX Foundry Container Logs
  - **docker-compose logs -f [compose-container-name]**
  - **e.g., log for data container**

```
Jim@JimsXPS MINGW64 ~/edgexfoundry/developer-scripts (master)
$ docker-compose logs -f data
Attaching to edgex-core-data
[36medgex-core-data   |[0m [2017-05-28 16:26:57.670] boot - 5  INFO [main] --- AnnotationConfigApplicationContext: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@4d76f3f8: startup date [Sun May 28
16:26:57 GMT 2017]; root of context hierarchy
[36medgex-core-data   |[0m [2017-05-28 16:26:59.438] boot - 5  INFO [main] --- AutowiredAnnotationBeanPostProcessor: JSR-330 'javax.inject.Inject' annotation found and supported for autowiring
[36medgex-core-data   |[0m [2017-05-28 16:26:59.675] boot - 5  INFO [main] --- PostProcessorRegistrationDelegate$BeanPostProcessorChecker: Bean 'configurationPropertiesRebinderAutoConfiguration' of type [class org.springframework.cloud
.autoconfigure.ConfigurationPropertiesRebinderAutoConfiguration$$EnhancerBySpringCGLIB$$8b45518f] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
[36medgex-core-data   |[0m
[36medgex-core-data   |[0m
[36medgex-core-data   |[0m
[36medgex-core-data   |[0m
[36medgex-core-data   |[0m /*******************************************************************************
[36medgex-core-data   |[0m  * Copyright 2016-17, Dell, Inc.  All Rights Reserved.
[36medgex-core-data   |[0m  *******************************************************************************/
[36medgex-core-data   |[0m
[36medgex-core-data   |[0m [2017-05-28 16:27:02.060] boot - 5  INFO [main] --- Application: The following profiles are active: docker
[36medgex-core-data   |[0m [2017-05-28 16:27:14.013] boot - 5  WARN [main] --- URLConfigurationSource: No URLs will be polled as dynamic configuration sources.
[36medgex-core-data   |[0m [2017-05-28 16:27:14.015] boot - 5  INFO [main] --- URLConfigurationSource: To enable URLs as dynamic configuration sources, define System property archaius.configurationSource.additionalUrls or make config.p
roperties available on classpath.
[36medgex-core-data   |[0m [2017-05-28 16:27:14.027] boot - 5  WARN [main] --- URLConfigurationSource: No URLs will be polled as dynamic configuration sources.
[36medgex-core-data   |[0m [2017-05-28 16:27:14.035] boot - 5  INFO [main] --- URLConfigurationSource: To enable URLs as dynamic configuration sources, define System property archaius.configurationSource.additionalUrls or make config.p
roperties available on classpath.
[36medgex-core-data   |[0m [2017-05-28 16:27:14.624] boot - 5  INFO [pool-1-thread-1] --- HeartBeat: Core data heart beat
[36medgex-core-data   |[0m [2017-05-28 16:27:14.841] boot - 5  INFO [main] --- Application: Started Application in 19.815 seconds (JVM running for 21.746)
[36medgex-core-data   |[0m This is the Core Data Micro Service.
[36medgex-core-data   |[0m [2017-05-28 16:32:14.614] boot - 5  INFO [pool-1-thread-1] --- HeartBeat: Core data heart beat
[36medgex-core-data   |[0m [2017-05-28 16:37:14.614] boot - 5  INFO [pool-1-thread-1] --- HeartBeat: Core data heart beat
```

상태 점검 (2/3)

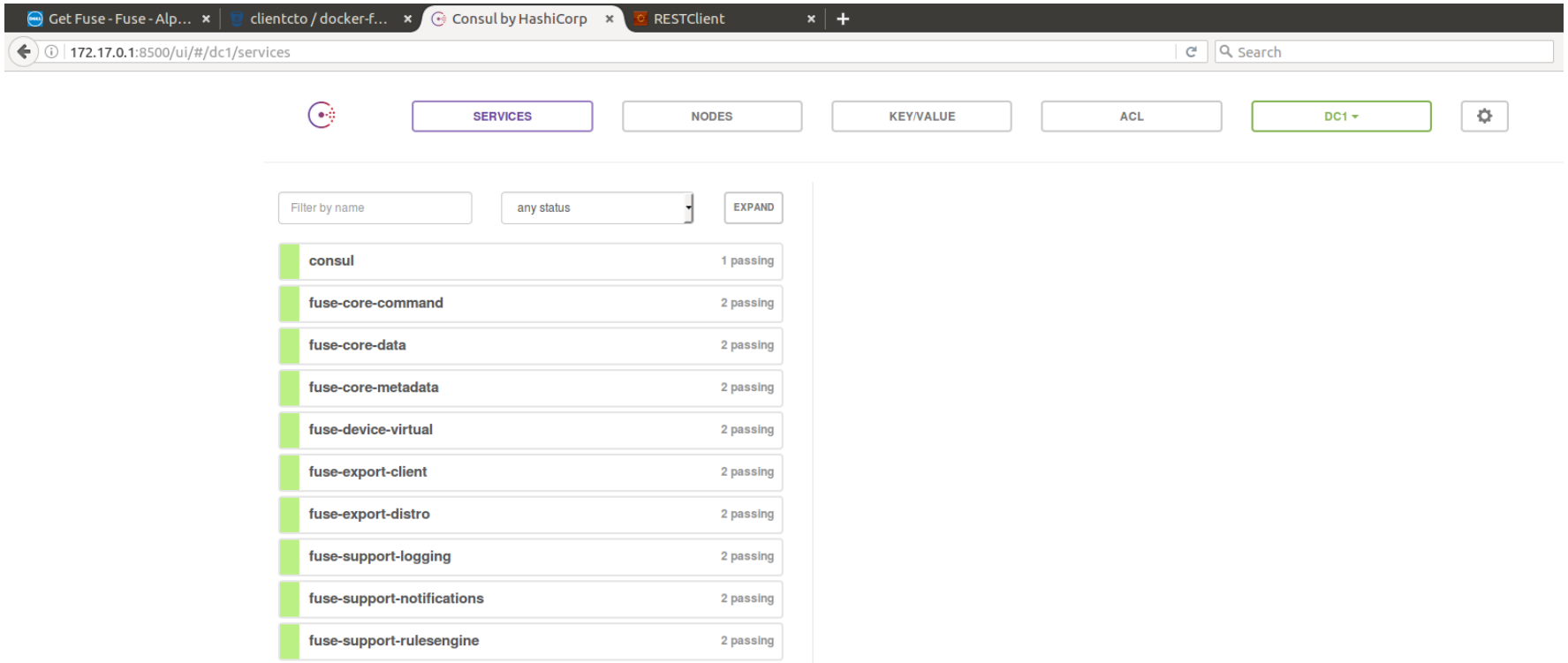
- Microservice Ping Check
  - "ping" any one of the microservices to check that it is running.



EdgeX Foundry Micro Service	Docker Compose Container	Container name	Port	Ping URL
Core Metadata	metadata	edgex-core-metadata	48081	http://[host]:48081/api/v1/ping
Core Data	data	edgex-core-data	48080	http://[host]:48080/api/v1/ping
Core Command	command	edgex-core-command	48082	http://[host]:48082/api/v1/ping
Support Logging	logging	edgex-support-logging	48061	http://[host]:48061/api/v1/ping
Support Notifications	notifications	edgex-support-notifications	48060	http://[host]:48060/api/v1/ping
Export Client	export-client	edgex-export-client	48071	http://[host]:48071/api/v1/ping
Export Distribution	export-distro	edgex-export-distro	48070	http://[host]:48070/api/v1/ping
Rules Engine	rulesengine	edgex-support-rulesengine	48075	http://[host]:48075/api/v1/ping
Virtual Device Service	device-virtual	edgex-device-virtual	49990	http://[host]:49990/api/v1/ping

상태 점검 (3/3)

- EdgeX Foundry Consul Registry
  - EdgeX Foundry uses the open source Consul project as its registry service
  - Find the Consul UI at **http://[host]:8500/ui**



Appendix

EdgeX 설치 및 실행
– 개발자 버전 –

(참고) EdgeX에 관한 자료는 EdgeX Foundry (https://edgexfoundry.org) 자료를 인용하였습니다.

기본 준비 사항

- Hardware
  - Memory: minimum of 4 GB
  - Hard drive space: minimum of 3 GB
  - OS:
    - Windows (ver 7 - 10)
    - Ubuntu Desktop (ver 14-16)
    - Ubuntu Server (ver 14)
    - Ubuntu Core (ver 16)
    - Mac OS X 10
- Software
  - Git, MongoDB, Java, Eclipse

EdgeX 코드 받기

- Download from github
  - https://github.com/edgexfoundry
  - git clone https://github.com/edgexfoundry/<library or microservice repos name>

```
$ git clone https://github.com/edgexfoundry/support-logging.git
Cloning into 'support-logging'...
remote: Counting objects: 60, done.
remote: Total 60 (delta 0), reused 0 (delta 0), pack-reused 60
Unpacking objects: 100% (60/60), done.
Checking connectivity... done.
```

Mongo 데이터베이스

- EdgeX Foundry's Mongo Database
  - EdgeX에서는 microservice에서 data나 metadata 저장을 위해 MongDB를 사용
    - core-data, core-metadata, support-rulesengine, supporting-logging (in some situations), among others
  - Initializing and running DB
    - developer-scripts repository의 스크립트 사용 가능

EdgeX Foundry in Eclipse

- Import the Project
  - Each of the EdgeX Foundry repositories containing source code is also an Eclipse project, specifically a Maven project
  - device-sdk-tools is not a Maven project
    - It should be imported using File → Import → General, Existing Project into Workspace
- Build and Install
  - Right click on the project and select Run As → Maven Install Running in Eclipse
  - Right click on the EdgeX Foundry microservice that you want to run and select Run As → Java Application