EDGE X FOUNDRY™

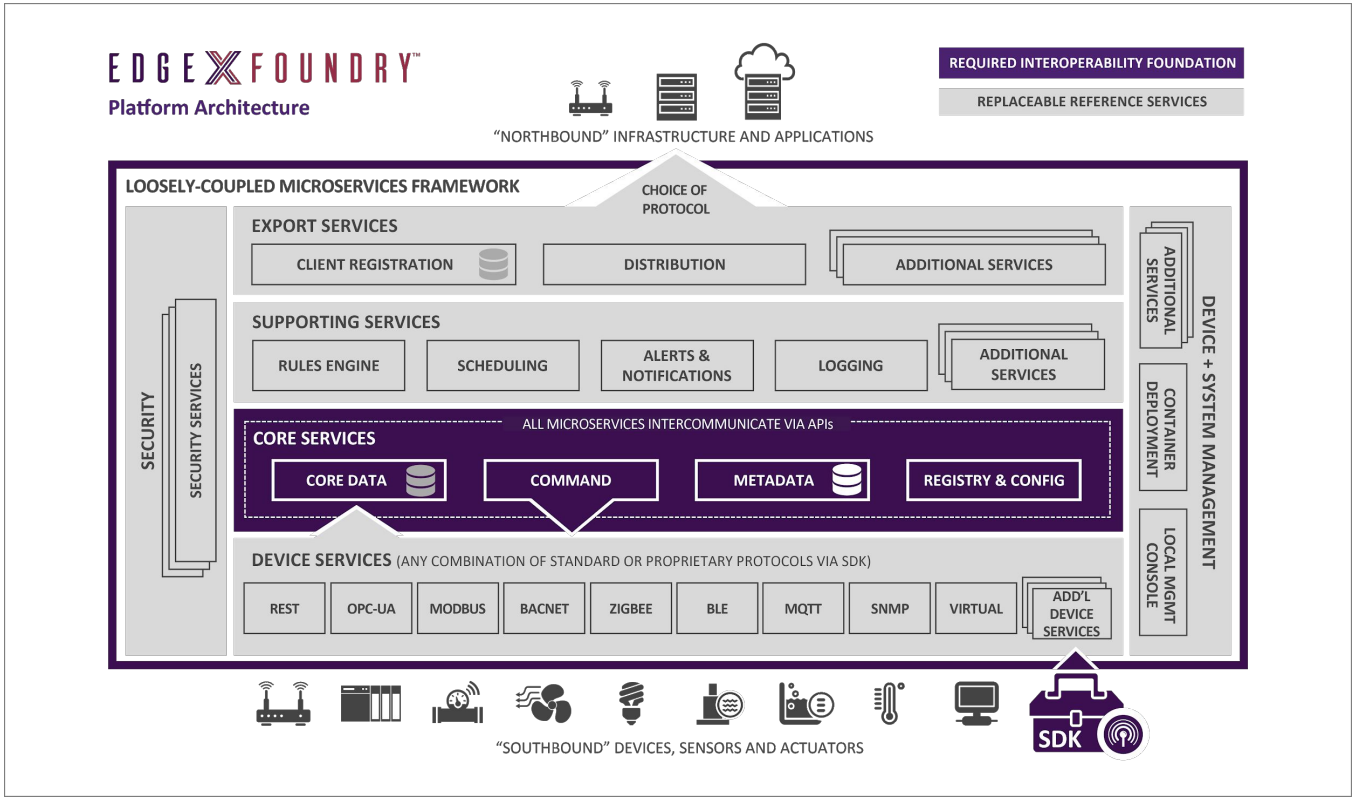
Building a Device Service using the Go SDK

Seoul F2F Technical Training

Tony Espy <espy@canonical.com>
Cloud Tsai <cloud@iotech.com>
Toby Mosby <tobias.mosby@intel.com>

April 30, 2019

EdgeX Foundry - Architecture



What's a Device Service?

- A device service (DS):
  - supports a specific device or class of devices/sensors
  - is a bridge that connects devices & sensors to EdgeX
  - provides REST API endpoints used by other EdgeX services
    - read data from devices/sensors
    - write data to devices/sensors
  - pushes device/sensor Events & Readings to Core Data
    - asynchronously (push)
    - on-demand (via REST calls)
    - scheduled (via AutoEvents)

What's a Device Profile?

- A Device Profile is a model in Core Metadata which:
  - represents a class of devices/sensors supported by a DS
  - defines some basic metadata (name, description, …)
  - defines a set of basic values that can be read/written
  - defines a set of commands for reading/writing values from a device/sensor
  - defines additional metadata used by Core Command

What's a Device Profile (continued)?

- Device profiles can be imported:
  - device-sdk-go
    - via YAML file import
    - read from the same directory as local configuration (/res)
  - Core Metadata:
    - upload YAML device profile files via a REST endpoint
    - import JSON device profile via REST endpoint

What's a Device Profile (continued)?

- A device profile has four sections:
  - Basic metadata (name, manufacturer/model, description, …)
  - Device Resources
  - Device Commands
  - Core Commands

What's a Device Profile (continued)?

- Device Resources and Device Commands sections defines the list of "commands" that are useable with the "device" REST endpoint:
  - `/device/{id}/{command}`
  - `/device/name/{name}/{command}`
- GET requests to these endpoints return an **Event** and one or more **Readings** (which hold device resource values)
  - ...and also trigger the **Event**/**Readings** to be pushed to Core Data

What's a Device Profile (continued)?

- PUT requests to these endpoints perform writes to the underlying device resource(s)

Device Resources

- A readable/writable named value on a device or sensor
- Supports a basic set of types:
  - string
  - bool
  - int8 | int16 | int32 | int64
  - uint8 | uint16 | uint32 | uint64
  - float32 | float64
    - encoded using base64 or C-style ("3.2165e+2")
  - binary
- Used to create a value descriptor object in Core Data

Device Commands

- Device commands allow aggregation of device resources
  - i.e. a single device command can read/write multiple device resources in a single REST* call
- Device commands defintions include lists of GET and SET commands called Resource Operations which reference Device Resources.

Core Commands

- The Core Command section defines commands that are usable with the Core Command "command" REST endpoint:
  - `/device/{id}/command/{command}:`
  - `/device/name/{name}/{command}`
- These commands also define:
  - expected values (i.e. value descriptor names)
  - expected REST response codes (e.g. 200, 404)
  - allowed parameter names (for writes)

Value Descriptors

- Value descriptors are Core Data objects which are created from a device profile's devices resources
- They define attributes of device resources (names & types)
- Value descriptor types are the same as Device Resource types
  - ex. int8, float32, binary, ...
- ...and are also used to describe parameters for SET commands

Example Profile - Simple-Device

name: "Simple-Device"

manufacturer: "Simple Corp."

model: "SP-01"

description: "Example of Simple Device"

.

.

.

Example Profile - Simple-Device (continued)

```
deviceResources:
    name: "SwitchButton"
    description: "Switch On/Off."
    properties:
      value:
        { type: "bool", readWrite: "RW" }
      units:
        { type: "String", readWrite: "R", defaultValue: "On/Off" }
    name: "Image"
    description: "Visual representation of Switch state."
    properties:
      value:
        { type: "binary", readWrite: "R" }
      units:
        { type: "string", readWrite: "R", defaultValue: "On/Off" }
```

Example Profile - Simple-Device (continued)

```
deviceCommands:
 -
    name: "Switch"
    get:
      - { operation: "get", object: "SwitchButton", property: "value", parameter: "Switch" }
    set:
      - { operation: "set", object: "SwitchButton", property: "value", parameter: "Switch" }
 -
    name: "Image"
    get:
     - { operation: "get", object: "Image", property: "value", parameter: "Image" }
```

Example Profile - Simple-Device (continued)

```
coreCommands:
 -
    name: "Switch"
    get:
      - { operation: "get", object: "SwitchButton", property: "value", parameter: "Switch" }
    set:
      - { operation: "set", object: "SwitchButton", property: "value", parameter: "Switch" }
 -
    name: "Image"
    get:
     - { operation: "get", object: "Image", property: "value", parameter: "Image" }
```

Creating Devices

- New devices can be created:
  - from local configuration file (configuration.toml)
  - from registry (aka consul) configuration
  - directly in Core Metadata via REST endpoint
  - via an SDK function call (`AddDevice`)
- Devices contain a map called Protocols which itself is a map of protocol specific properties. Ex.

```
Protocols [ serial: [baud:9600, bits:7, port: com1, …] ]
```

AutoEvents

- Each device has a list of zero or more AutoEvents
- An AutoEvent is an object used to schedule a device service to push a Reading to Core Data on a scheduled basis
- AutoEvents are defined by:
  - a frequency (ex. 1s, 2m, 3h, …)
  - a DeviceCommand
  - OnChange flag

A New Go-based Device Service - Preparation

- The following are prerequisites for developing a new Go-based device service:
  - go 1.11
  - go-mod-core-contracts
  - device-sdk-go

```
$ go get github.com/edgexfoundry/go-mod-core-contracts
$ go get github.com/edgexfoundry/device-sdk-go
```

Overview of device-sdk-go

- The SDK provides all of the boilerplate code for an EdgeX device service to manage devices and sensors
- This includes:
  - configuration
  - registry integration
  - integration with core & support services
  - auto-events
  - asynchronous readings
  - REST endpoints
  - device profile imports

pkg/models - ProtocolDriver

- A Go interface which provides an API to facilitate a device service's protocol-specific logic.
- This interface defines the following methods:
  - `Initialize`
  - `DisconnectDevice`
  - `HandleReadCommands`
  - `HandleWriteCommands`
  - `Stop`

pkg/models - ProtocolDriver (continued)

- `Initialize` - key entrypoint for device services to perform:
  - protocol-specific initialization
  - start threads to handle device management
- `Stop` - entrypoint to handle service shutdown
- `DisconnectDevice` - handle device removal
- `HandleRead/WriteCommands`
  - called in response to REST calls and AutoEvents

pkg/models - CommandValue

- `CommmandValue` is used to pass protocol specific reading from a `ProtocolDriver` implementation to the SDK (which then converts the values to a string value saved in a `Reading`)
  - `ValueType` - an enum which indicates what type is being returned
  - `NumericValue` - an array of bytes that holds the underlying bytes (Big Endian) of a numeric value
  - `BinaryValue` - an array of bytes that represents a binary reading

pkg/startup - Bootstrap

- Provides optional startup `Bootstrap` functionality:
  - command-line processing
  - configuration loading
  - starts main device service listener

Examples

- device-simple
  https://github.com/edgexfoundry/device-sdk-go/tree/master/example
- device-random
  https://github.com/edgexfoundry/device-random
- device-mqtt
  https://github.com/edgexfoundry/device-mqtt

Q&A