Movable Device Provisioning

EdgeX Foundry – Geneva Release
Use Case

• Device moves from Device Service 1 to Device Service 2
• Example: a wearable device on a person that moves from building to building
• How best to define an API to support this?
• Note – At the time of writing, design on Geneva’s “Provision Watcher” API is not finalized
• Assumption – Transport shown in these use cases is REST API. Review with an eye toward possibility of pub/sub.
Option 1: Implicit Re-assignment

POST /api/v2/device

{AddDeviceRequest}

Does device exist?

Yes, it exists (unique name check)

Re-assign to new device service

OK

HTTP 200 -- OK

{AddDeviceResponse}

** Problematic if this interaction fails. See slide 6 **

PUT /api/v2/device

{RemoveDeviceRequest}

HTTP 200 -- OK

{RemoveDeviceResponse}
Option 1: Pros / Cons

• Pros
Less effort for clients to integrate. In fact this can be done with minor changes to the current API
Reconciliation of device to service association centralized in core-metadata

• Cons
Re-assignment of device to new service is implicit via dual-purpose endpoint
Less granular observability
Notification to Device Service 1 could fail, orphaning device (possible issue today)
Knowledge of how to interpret a 409 conflict on the Device Service’s behalf is in Core-Metadata. Doesn’t seem like the right responsibility.
Option 2: Explicit Re-assignment

**Device Service 2**

- POST /api/v2/device
  - {AddDeviceRequest}
  - HTTP 409 -- Conflict
  - {AddDeviceResponse}
- GET /api/v2/device/biometric123
  - **Used to compare new service name with that currently on the device. If different, re-assign**
  - {Device}
- POST /api/v2/device/reassign
  - {ReassignDeviceRequest}
- HTTP 200 -- OK
  - {ReassignDeviceResponse}

**Core-Metadata**

- Does device exist?
  - Yes, it exists (unique name check)

**DB**

- Load the device record
- Re-assign to new device service
- OK
- PUT /api/v2/device
  - {RemoveDeviceRequest}
  - HTTP 200 -- OK
  - {RemoveDeviceResponse}

**Device Service 1**

**Problematic if this interaction fails. See next slide**

---

*Current interaction fails, see next slide for alternative approach.*
Option 2: Pros/Cons

• Pros
Explicit differentiation between “AddDevice” and “ReassignDevice”
  b/c of need to handle 409
Following from above, more granular observability
Separation of concerns – from core-metadata’s perspective, does a conflict always
  result in a device reassignment? Device Service can make this decision.

• Cons
Interaction between DeviceService and Core-Metadata is a bit more complex
  due to explicit contract
Doesn’t resolve possibility for orphaned device due to failed notification to DS1
Option 2a: Explicit Re-assignment

**Device Service 2**

POST /api/v2/device

{AddDeviceRequest}

HTTP 409 -- Conflict

{AddDeviceResponse}

GET /api/v2/device/biometric123

{Device}

**Device model wraps current addressable**

**Device Service 1**

POST /api/v2/device/reassign

{ReassignDeviceRequest}

HTTP 200 -- OK

{ReassignDeviceResponse}

**Core-Metadata**

Does device exist?

Yes, it exists (unique name check)

Load the device record

Re-assign to new device service

OK

**Device model wraps current addressable**

PUT /api/v2/device

{UpdateDeviceRequest}

HTTP 200 -- OK

{UpdateDeviceResponse}

**Core-Metadata is kept in sync and Device Service 1 cleans its device cache upon receipt of 200 OK**
Option 2a: Pros/Cons

• Pros
See Option 2 “pros” previous slide
Mitigation of orphaned devices by ensuring reassigned device is cleared from original device service before being re-assigned
Further delegation of how to interpret a conflict (409) status to the Device Service

• Cons
Interaction between DeviceService and Core-Metadata is a bit more complex due to explicit contract
Development overhead to introduce DS to DS communication, expands requisite testing