iot
eclipse.org

Manage your devices with Lightweight M2M and connect them to your cloud

EclipseCON EU 2016

Agenda

CoAP & Lightweight M2M

Demo

Eclipse Leshan

Hands-on!

Eclipse Hono

Going further

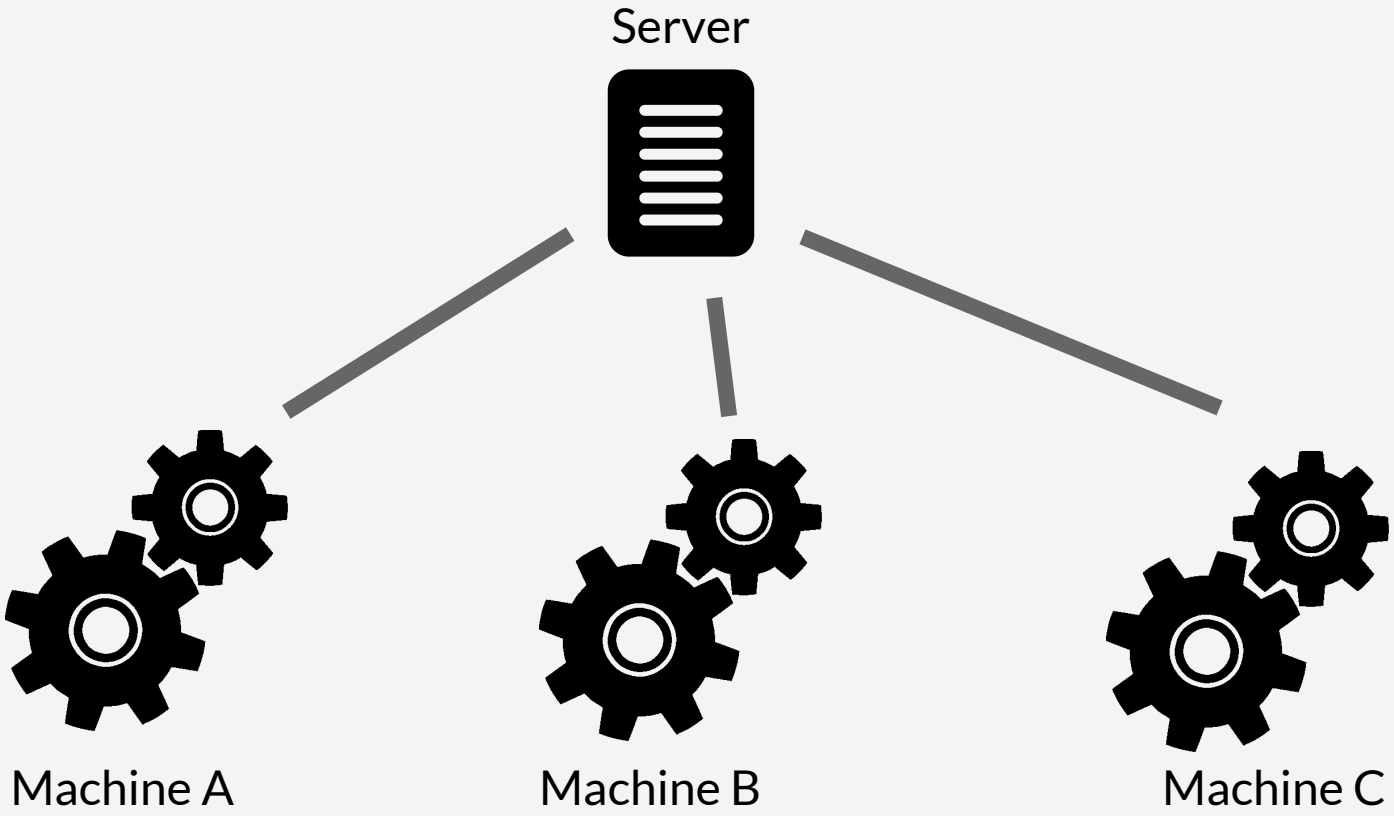
Follow the slides

https://goo.gl/uzXThJ

Your devoted presenters

From M2M to Web-of-Things

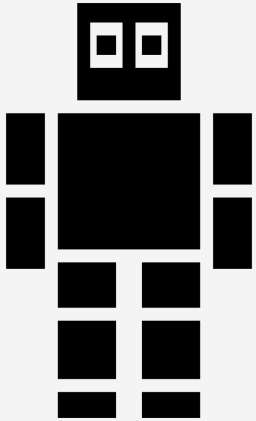
Machine-to-Machine

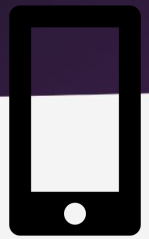


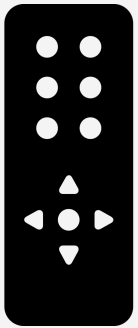
Conquer the last mile

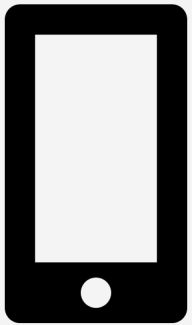
- Low-power networks plugged to the Internet
- 6LoWPAN
- Bluetooth Smart 4.2
- Thread
- LWPA (LoraWAN, LTE-MTC,...)
- IPv6 MTU: 1280 bytes, 6LowPAN: ~100 bytes
- TCP, HTTP,MQTT doesn't fit

Internet of Things

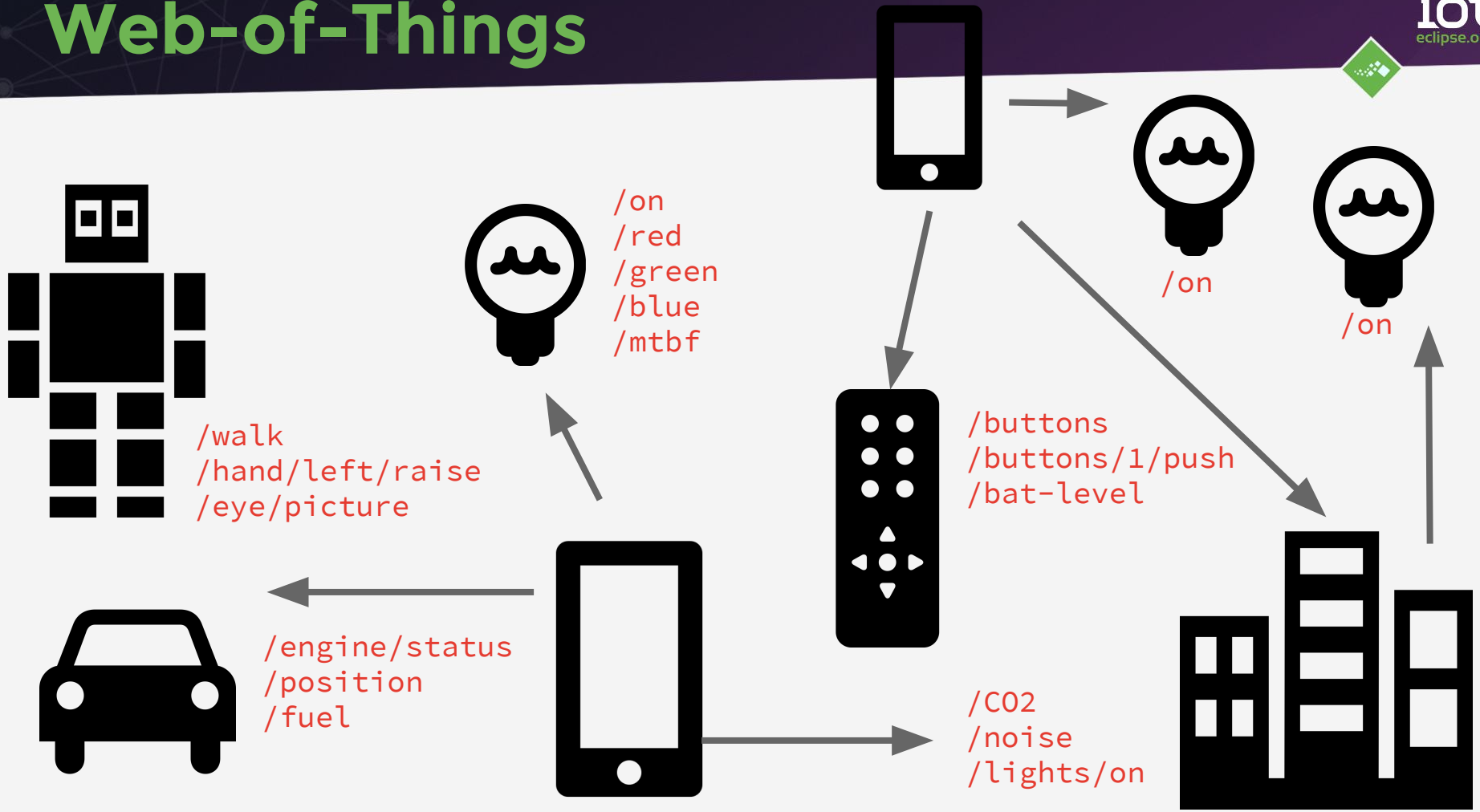








Web-of-Things



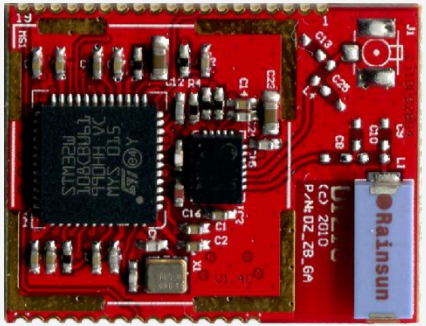
iot
eclipse.org

CoAP

Constrained Application Protocol

CoAP: a new protocol for IoT

Class 1 devices
~100KiB Flash
~10KiB RAM
~$1



Low-power networks
<100Bytes packets

RFC 7252: CoAP

- RESTful protocol designed from scratch
  - URIs, Internet Media Types
  - GET, POST, PUT, DELETE
- Transparent mapping to HTTP
- Additional features for M2M scenarios
  - Observe

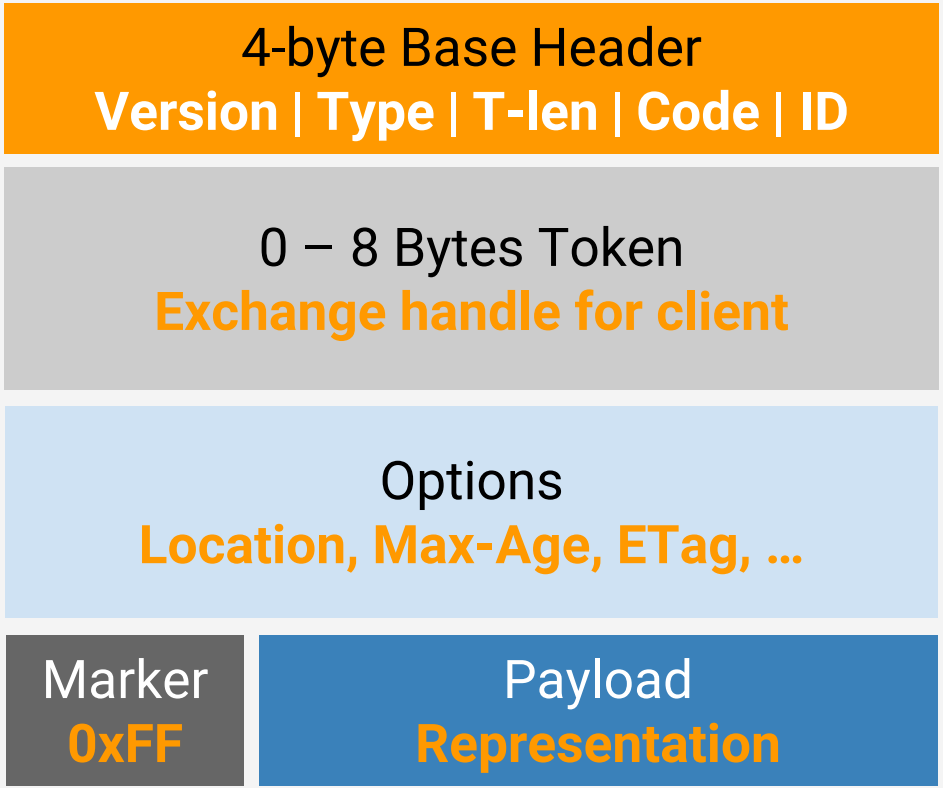
CoAP

Binary protocol

- Low parsing complexity
- Small message size

Options

- Binary HTTP-like headers



Device Management

Operate, monitor, upgrade fleets

Device Management

Secure, monitor, manage a fleet of devices

Configure the device

Update the firmware (and maybe the app)

Monitor and gather connectivity statistics

Device Management

You don't know yet what hardware will power your IoT projects on the field,

But you MUST be able to do device management in a consistent way without vendor lock

iot
eclipse.org

OMA Lightweight M2M

An API on top of CoAP

Lightweight M2M

REST API for:

Security provisioning

Connectivity configuration, monitoring, statistics

Location

Firmware Upgrade

Software management

Error reporting

LwM2M API URLs

`/{object}/{instance}/{resource}`

Examples:

"**`/6/0`**" the whole location object (binary record)

"**`/6/0/1`**" only the longitude (degree)

Object Name	ID	Multiple Instances?	Description
LWM2M Security	0	Yes	This LWM2M Object provides the keying material of a LWM2M Client appropriate to access a specified LWM2M Server.
LWM2M Server	1	Yes	This LWM2M objects provides the data related to a LWM2M server.
Access Control	2	Yes	Access Control Object is used to check whether the LWM2M Server has access right for performing an operation.
Device	3	No	This LWM2M Object provides a range of device related information which can be queried by the LWM2M Server, and a device reboot and factory reset function.
Connectivity Monitoring	4	No	This LWM2M objects enables monitoring of parameters related to network connectivity.
Firmware	5	No	This Object includes installing firmware package, updating firmware, and performing actions after updating firmware.
Location	6	No	The GPS location of the device.
Connectivity Statistics	7	No	This LWM2M Objects enables client to collect statistical information and enables the LWM2M Server to retrieve these information, set the collection duration and reset the statistical parameters.

Example: Object Device

- Manufacturer
- Model number
- Serial number
- Firmware version
- Reboot
- Factory reset
- Power sources
- Power V/A
- Battery level
- Memory free
- Error code
- Current time
- UTC offset
- Timezone

You can define your own objects and register with the OMA

IPSO Alliance Smart Objects:

accelerometer, temperature, sensors,...

Object	Object ID	Multiple Instances?
IPSO Digital Input	3200	Yes
IPSO Digital Output	3201	Yes
IPSO Analogue Input	3202	Yes
IPSO Analogue Output	3203	Yes
IPSO Generic Sensor	3300	Yes
IPSO Illuminance Sensor	3301	Yes
IPSO Presence Sensor	3302	Yes
IPSO Temperature Sensor	3303	Yes
IPSO Humidity Sensor	3304	Yes
IPSO Power Measurement	3305	Yes
IPSO Actuation	3306	Yes
IPSO Set Point	3308	Yes
IPSO Load Control	3310	Yes
IPSO Light Control	3311	Yes
IPSO Power Control	3312	Yes
IPSO Accelerometer	3313	Yes
IPSO Magnetometer	3314	Yes
IPSO Barometer	3315	Yes

Type	Object	Object ID
Common Template Sensors	Voltage	3316
	Current	3317
	Frequency	3318
	Depth	3319
	Percentage	3320
	Altitude	3321
	Load	3322
	Pressure	3323
	Loudness	3324
	Concentration	3325
	Acidity	3326
	Conductivity	3327
	Power	3328
	Power Factor	3329
	Rate	3346
	Distance	3330
Special Template Sensors	Energy	3331
	Direction	3332
	Time	3333
	Gyrometer	3334
	Color	3335
	GPS Location	3336
Actuators	Positioner	3337
	Buzzer	3338
	Audio Clip	3339
	Timer	3340
	Addressable Text Display	3341
Controls	On/Off Switch	3342
	Push Button	3347
	Level Control	3343
	Up/Down Control	3344
	Multistate Selector	3348
	Multiple Axis Joystick	3345

Demo!

Security with LWM2M

Authentication & Encryption

Based on DTLS 1.2 (TLS for Datagrams)

Focus on AES & Elliptic Curve Cryptography (ECC)

AES Hardware acceleration in IoT oriented SoC

Works on Low Power networks (~100bytes MTU)

TLS_PSK_WITH_AES_128_CCM_8

Pre-Shared-Key:
password for session authentication

AES 128bits (or 256) - Counter CBC Mode:
encryption and integrity (AEAD cipher)
8 bytes for integrity in place of CCM usual 16

What? :)

PSK: No certificates, just password

CCM8: compactness

Full DTLS-PSK-CCM8 handshake in ~1030b

Ex: HTTPS TLS handshake ~6kbytes

More security: TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8

ECDHE: Perfect Forward Secrecy (PFS)

Someone rob your private key: he can't decrypt past communications

ECDSA: use public key in place of password

You can use X.509 certificates (like HTTPS)

At Scale

You will have a fleet of device

They need secrets (key, password, etc..)

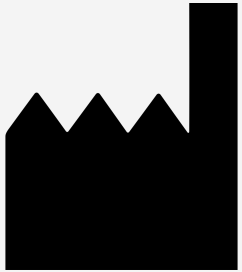
Unique across devices

You need to be able to change those secrets

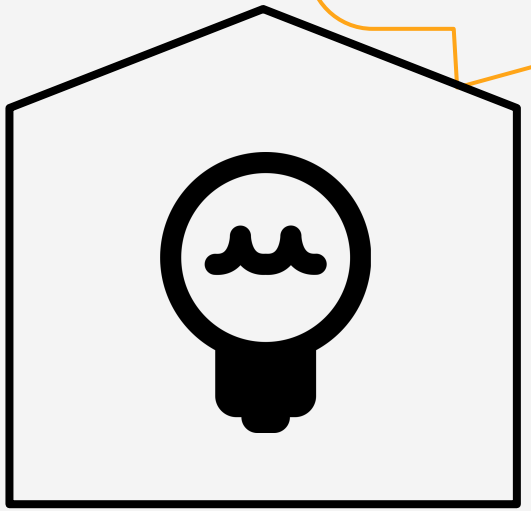
You will probably don't trust your factory

LwM2M Bootstrap

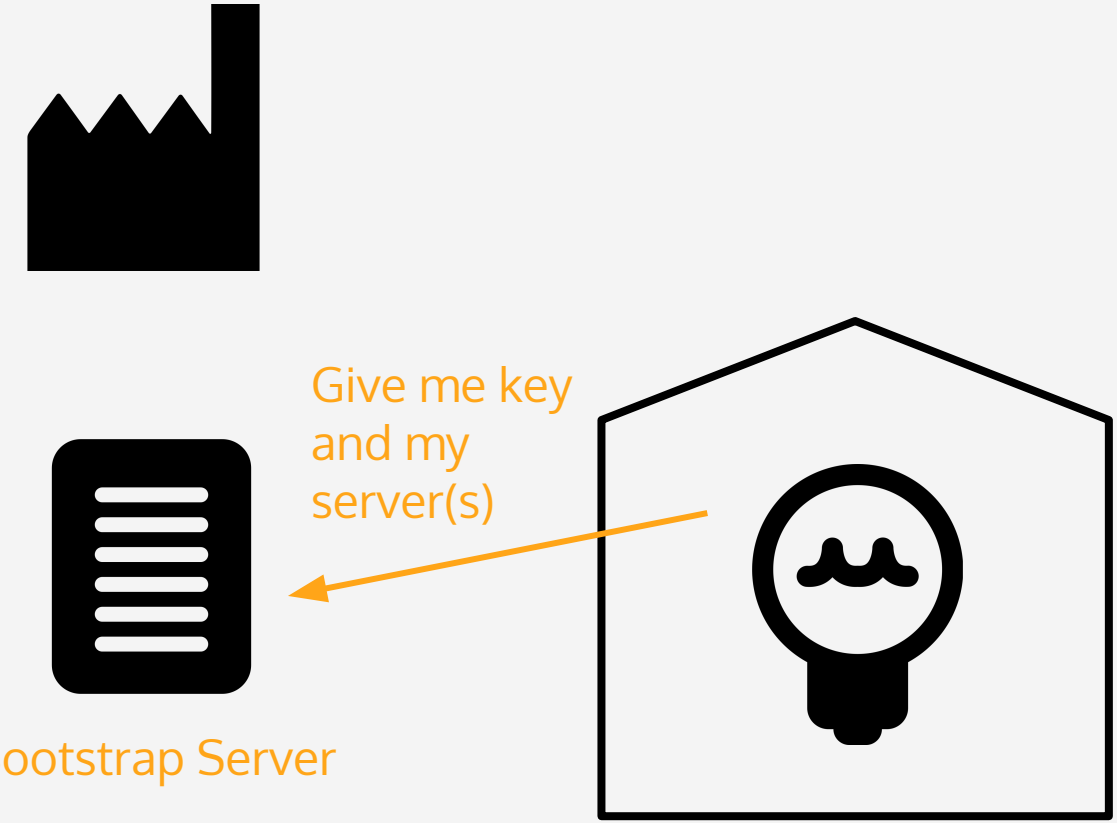
LwM2M Bootstrap



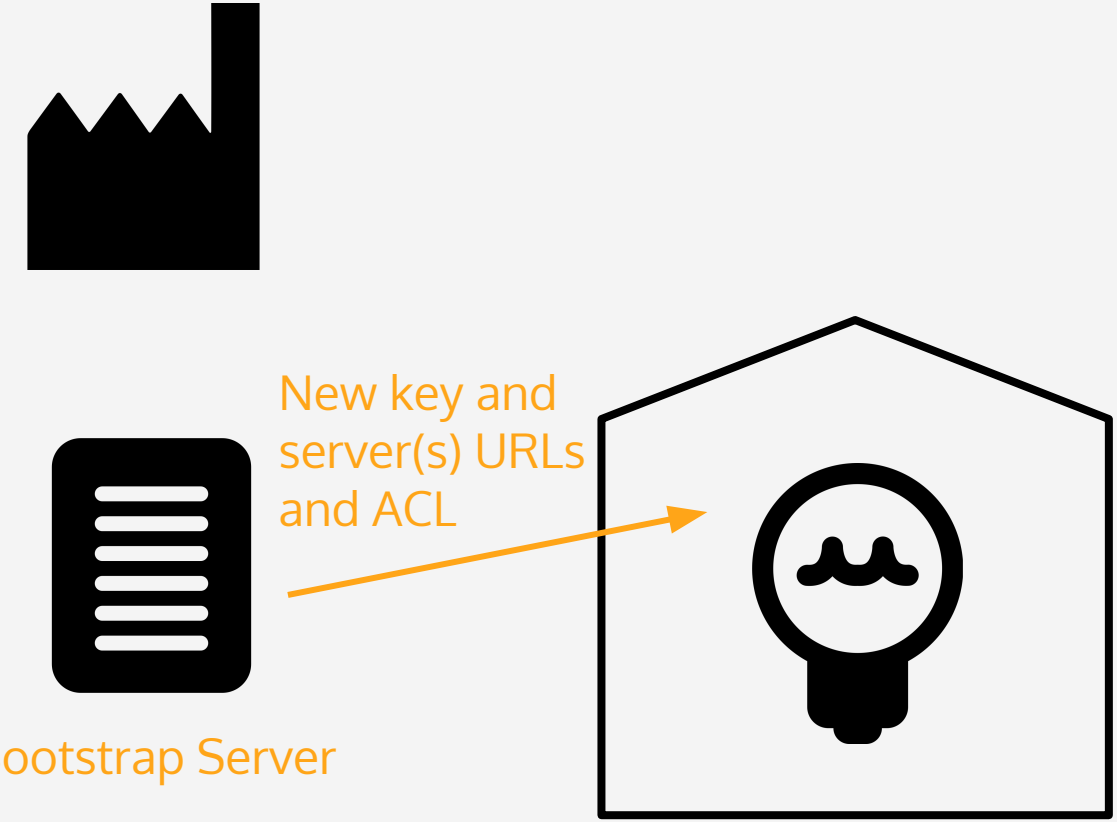
I only have bootstrap credentials or I can't reach final server



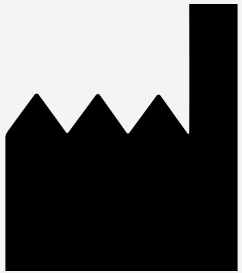
LwM2M Bootstrap

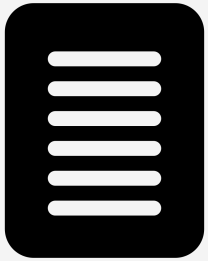


LwM2M Bootstrap

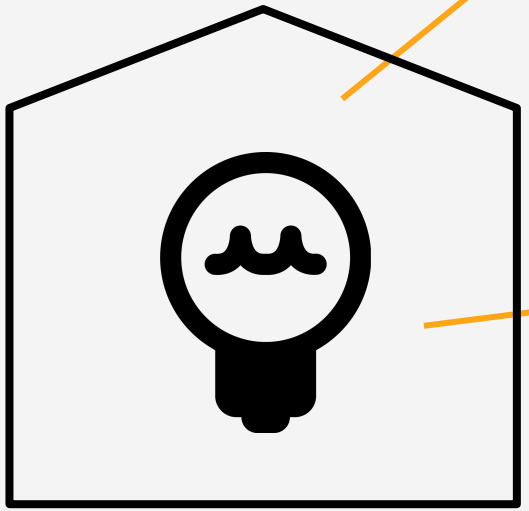


LwM2M Bootstrap

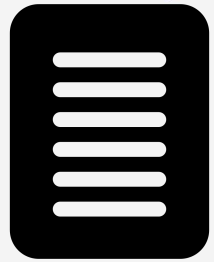




Bootstrap Server

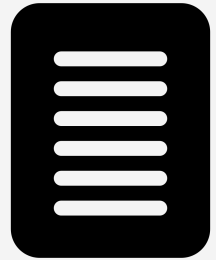


Registration



Home Automation Server

Registration



Device Manag. Server

iot
eclipse.org

Eclipse Leshan

LESHAN

Leshan

Java library for implementing servers & clients

Friendly for any Java developer

Simple (no framework, few dependencies)

But also a Web UI for discovering and testing

Build using “mvn install”

Based on Eclipse Californium and Scandium

Public Sandbox

http://leshan.eclipse.org

Bleeding edge: deployed on master commit

IPv4 and IPv6

Press “CoAP messages” for low-level traces

Leshan Modules

Leshan-core	commons elements.
Leshan-server-core	server lwm2m logic.
Leshan-server-cf	server implementation based on californium.
Leshan-client-core	client lwm2m logic.
Leshan-client-cf	client implementation based on californium.
Leshan-all	every previous modules in 1 jar.
Leshan-client-demo	a simple demo client.
Leshan-server-demo	a lwm2m demo server with a web UI.
Leshan-bsserver-demo	a bootstrap demo server with a web UI.
Leshan-integration-tests	integration automatic tests.

iot
eclipse.org

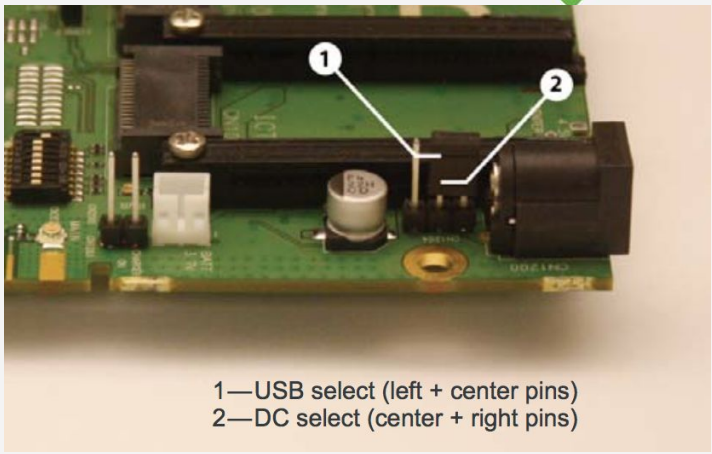
Hands-on!

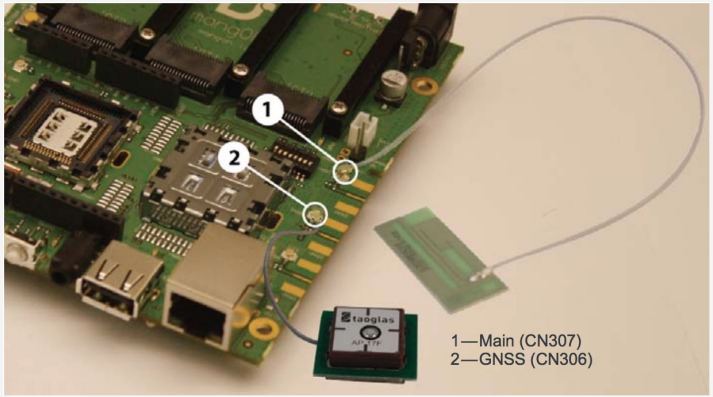
Getting started

Set up your mangOH board:

- Select DC power source (move jumper if needed)
- Connect the antenna (main)
- Connect the DC power supply
- Connect the board to your laptop using the micro-USB cable provided

Windows only: please install the USM/ECM driver (from the USB-stick) to be able to connect to the board (Ethernet over USB)





Getting started

Connect to your mangOH board:

- SSH to your board: root@192.168.2.2

  `ssh root@192.168.2.2` for linux/mac - PuTTY for Windows
- Once connected, print the device info:

  `cm info`
- Start the cellular data connection:

  `cm data connect&`

  (you will need to run it again later if the connection is lost)
- Wait a few seconds and check the connection:

  `cm data`

You're all set with the board, let's start coding!

Getting started

- Get the tutorial projects:
  https://github.com/msangoi/eceu2016-tutorial or from the USB-stick

- Launch Eclipse and import the projects:
  File > Import... > Existing projects into workspace

Step 1: run a basic client

1. Complete the code (project *client-step1*)

   Use the *LeshanClientBuilder* to instantiate a client and start it.

2. Create a runnable jar. On the project:
   - Export > Java > Runnable Jar file
   - Select the launch configuration *client-step1*
   - Enter a destination for your file
   - Check “Extract required libraries into generated JAR”
   - Finish (and ignore warnings)
3. Copy the generated jar to the mangOH board:

   `scp client.jar root@192.168.2.2:/home/root` for linux - WinSCP for Windows

4. Run the client:

   On the mangOH board: `./ejdk/bin/java -jar client.jar`

   With the default configuration, the client registers to the eclipse demo server: *coap://leshan.eclipse.org:5683*

5. Find your client on the demo server: http://leshan.eclipse.org

Step 2: initialize custom objects

1. Complete the code (project *client-step2*):
   - Implement your custom Device object (*MyDevice*)
   - Use the *ObjectsInitializer* to initialize an instance of MyDevice

2. Export the client as a jar file, copy it to the board and run it.

Note: For this step and the next ones, you can keep on using the eclipse demo server (if your cellular connection is ok) or you can use a local server running on your machine.

To start a demo server on your laptop, run the server-demo jar (from the USB-stick):

```
java -jar leshan-server-demo.jar
```

From the mangOH board, your laptop address is 192.168.2.3.

In your client java code, change the LWM2M server URI in the Security Object:

```
Security.noSec("coap://192.168.2.3:5683", 123)
```

Step 3: observe accelerometer data

1. Complete the code (project *client-step3*)
   - Implement the Accelerometer object (X, Y, Z values)
   - Update X,Y,Z periodically (e.g. every 500 ms)
   - Notify the value changes so that the client sends notifications to the server if the Accelerometer is observed.

2. Export the client as a jar file, copy it to the board and run it.

3. Start observing the Accelerometer object from the demo server.

Step 4: server implementation

1. Complete the code (project *server-step4*)
   - Instantiate a server using the *LeshanServerBuilder.*
   - Add a listener to the *ObservationRegistry* to be notified with new notification values.
   - Add a listener the the ClientRegistry to start an observation on the Accelerometer object (if supported).
   - Start the server

2. Run your last version of the client (or build a runnable jar from the *client-step3-complete* project)
   - You may need to modify the client code to register to the server running on your laptop (coap://192.168.2.3:5683)

Step 5 (bonus): PSK security

Client side

- Update the code from project *client-step2-complete.*
- Initialize the Security object with an instance providing security information for Pre-Shared Key mode:
    - The LWM2M server URI used to populate the Security object must use the "*coaps*" scheme and port *5684* (default secure port)
    - define a psk identity (unique)
    - define a key (hexadecimal string)

Server side - Using the Eclipse Demo server (http://leshan.eclipse.org or local)

- In the *Security* tab, enter the security configuration for your client.
- Register your client to the eclipse demo server.

iot
eclipse.org

How many technologies do you need to master to communicate with all things connected to the IoT?

iot
eclipse.org

Just one! Eclipse Hono.

(With a little help from the Eclipse IoT community)

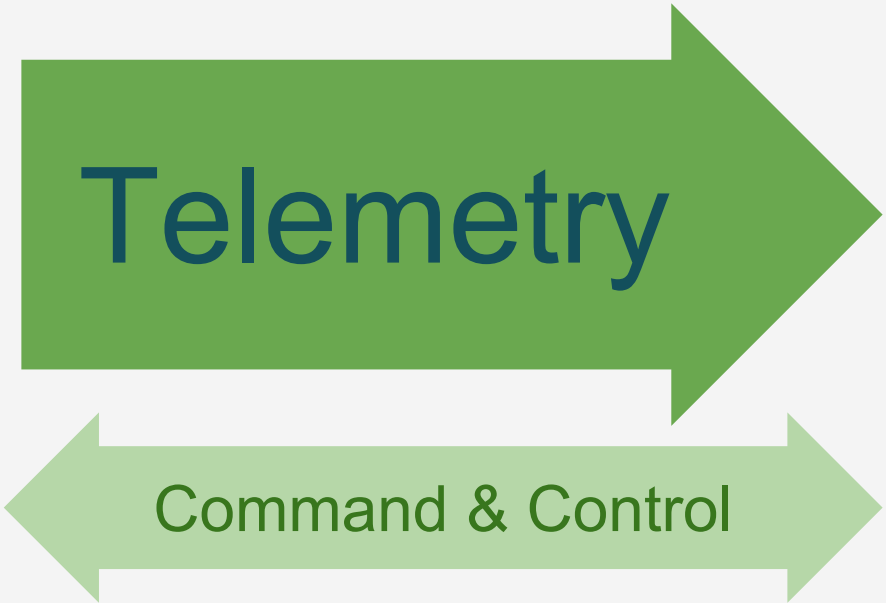
What is it good for?

Eclipse Hono provides a **uniform API** for interacting with **millions of devices** connected to the cloud via **arbitrary protocols**.

HONO

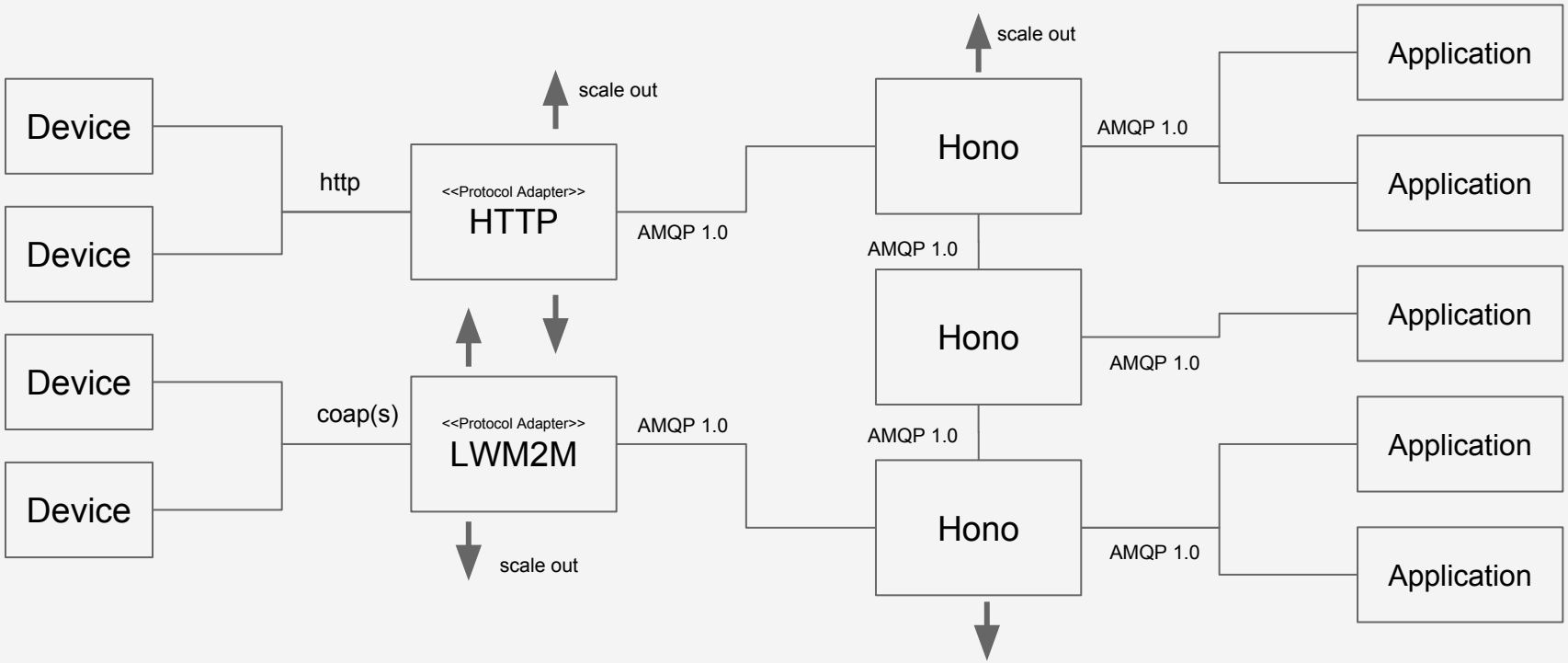
Characteristics

optimized for throughput
scale-out with #messages



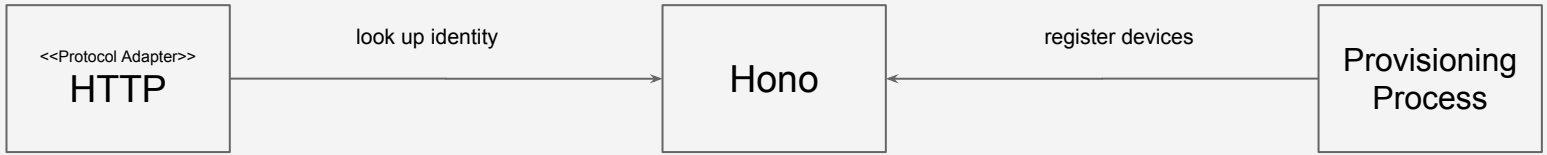
optimized for reliability
scale-out with #devices

General Concepts



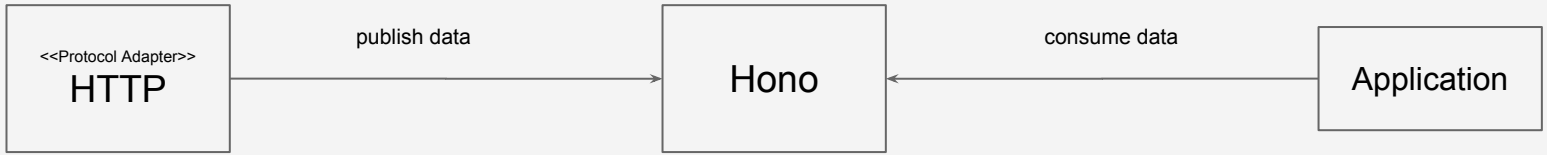
Arbitrary Protocol Adapters possible. We are working on HTTP, MQTT and LWM2M.

Registration API



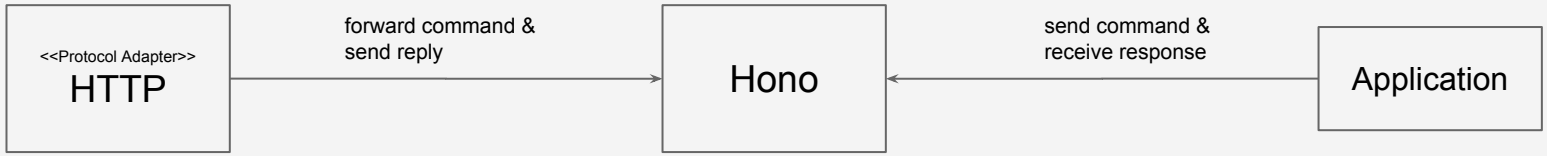
- Each device is registered with a *logical* ID scoped to a *tenant*.
- Optionally, additional key/value pairs can be registered for a device, e.g. a Pre-shared Key used authenticating the device as part of TLS
- Provisioning Process registers device identities with Hono.
- Protocol Adapters look up logical (Hono) identity by (technical) keys.

Telemetry API



- Protocol Adapters multiplex downstream data for devices of same *tenant*.
- Applications consume data for one or more tenants.

Command & Control API



- Applications send commands to devices.
- Hono brokers between the application and the protocol adapter the device is connected to.
- Protocol Adapters forward outbound messages to devices when they are connected.
- **Not implemented yet.**

Step 0

Prepare Environment

Download Tools

Point your browser to ftp://10.66.0.xxx (this is the anonymous FTP server on my laptop)

Download the following files

- `lc.jar` - the leshan demo client for simulating a LWM2M device
- `objectdefinitions.zip` - contains additional LWM2M Object definitions

Extract `objectdefinitions.zip` to a folder of your choice. It contains XML files describing LWM2M object types and their resources.

Start up the Hono Back End

We will start up an ensemble of services comprising the Hono back end using Docker Compose:

- Hono Server
- Qpid Dispatch Router
- Hono HTTP Adapter

Devices connect to the HTTP protocol adapter.

Applications connect to the Dispatch Router.

Later on we will add a LWM2M Protocol Adapter to which our LWM2M devices will connect via coap(s).

Steps to perform from command line

```
1.  cd $HONO_HOME/example/target/hono
2.  docker-compose up -d
```

Open http://your_docker_host:8080/status in a browser to verify startup. You should get a JSON object containing some diagnostic info. In particular, it should contain

```
"connected": true
```

Use the following command to shut down Hono

```
docker-compose down
```

Step 1

Use Hono as Leshan's SecurityRegistry

What we'll do

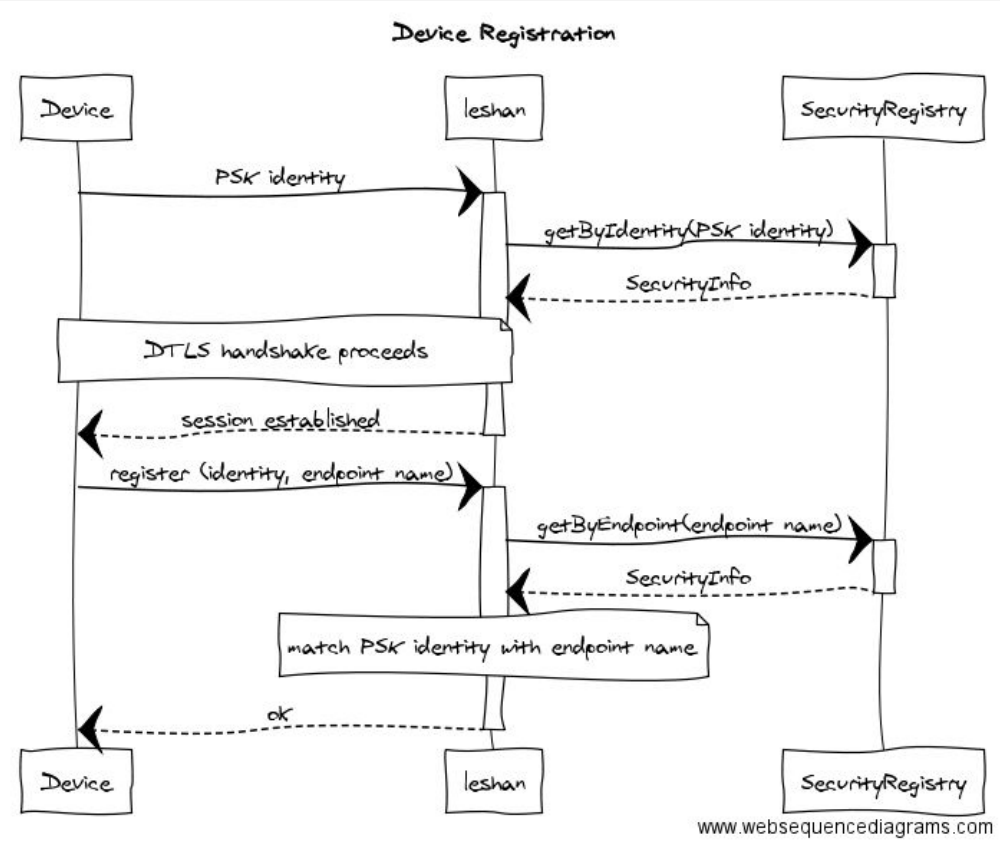
In this step we will

- configure leshan to use a custom `SecurityRegistry` which uses Hono for managing our device's security information
- start up the LWM2M Protocol Adapter
- Add security info for our device
- Start up a LWM2M client that registers with the adapter
- *observe* resources on the device for changes

LWM2M Register Sequence

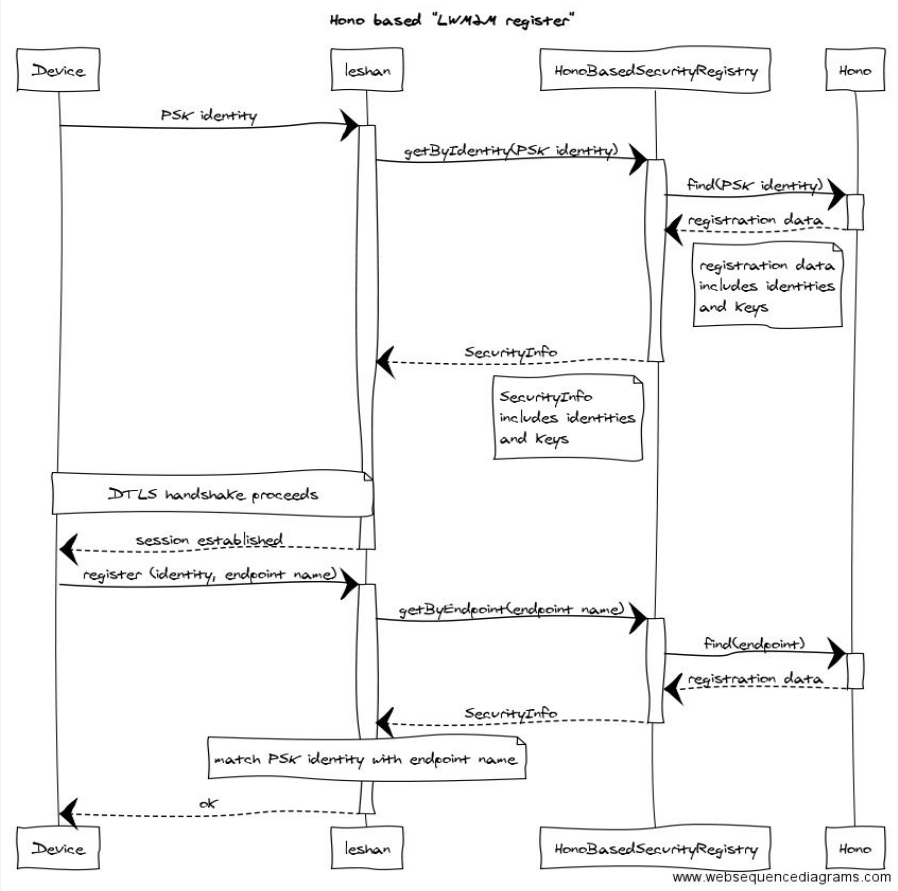
Device is authenticated as part of DTLS handshake using a Pre-shared Key (PSK) based cipher suite.

leshan checks whether client's *technical* identity corresponds to the *logical* endpoint name submitted as part of the `register` CoAP request.



Use Hono as SecurityRegistry

We will configure leshan to use the `HonoBasedSecurityRegistry` class which accesses Hono's Registration API to retrieve device registration data from Hono.



Start up LWM2M Adapter

1. Switch to branch `step1` in the Hono source folder, e.g. from the command line

   ```
   $> cd $HONO_HOME
   $> git checkout step1
   ```

   or use the corresponding means in your IDE

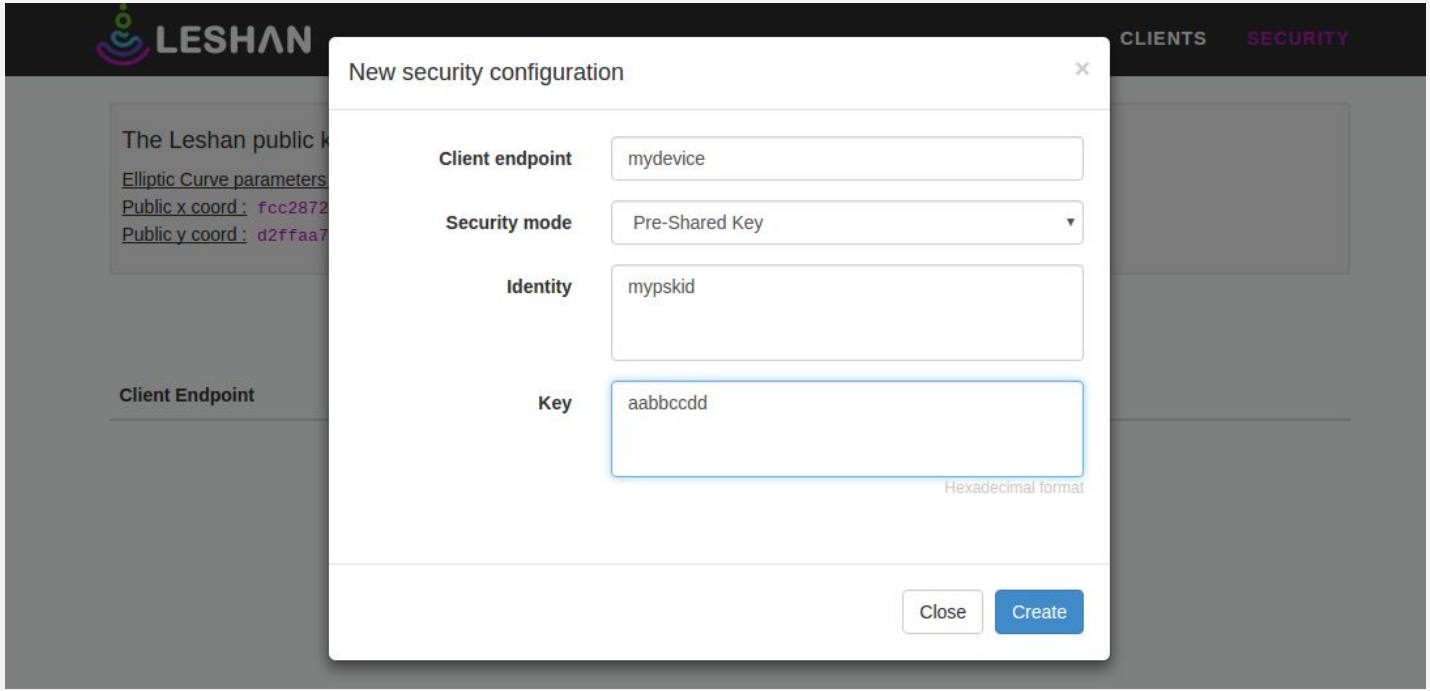
2. Start up LWM2M adapter

   ```
   $> cd $HONO_HOME/adapters/lwm2m
   $> mvn spring-boot:run -Dhono.client.host=your_docker_host
   ```

3. Read through the log and see that the adapter connects to Hono and uses our custom `HonoBasedSecurityRegistry` implementation

Add Device Security Info

1. Open web browser, go to `http://localhost:8090/#/security`
2. Add security info for your device



Add Device Security Info

Check that device has been registered with Hono using HTTP protocol adapter

```
$> curl -i -X POST -d ep=mydevice \
$> http://your_docker_host:8080/registration/DEFAULT_TENANT/find
```

or, using HTTPie

```
$> http -f POST http://your_docker_host:8080/registration/DEFAULT_TENANT/find  \
$> ep=mydevice
```

cURL for Windows: https://curl.haxx.se/dlwiz/?type=bin

HTTPie: https://httpie.org

Start up Device

Download leshan client from

- my (local) FTP server: ftp://10.66.0.xxx or
- my Google Drive: https://drive.google.com/open?id=0B8o-KHF-_cPKMFdLZjJaOFFRVWc

Run leshan client (from your folder)

```
$> java -jar lc.jar -n mydevice -i mypskid -p aabbccdd
```

Check that device has registered as LWM2M client

1. Open web browser, go to http://localhost:8090/#/clients
2. Play around with the device :-)

Observe Location

Observe Device Location

1. On device's detail page, click the `Observe` button next to `Instance 0` of the `Location` section
2. See how current Location is retrieved from device
3. Go to terminal in which leshan client is running and press any one of keys `W`, `A`, `S` or `D` followed by `CR`. This shifts the position north, west, south or east and sends a notification to the LWM2M server.
4. Go back to web browser and see how the values of the `Location` object have been updated
5. Play around with the observed `Location` resource

Stop device

1. Hit `ctrl-c` in terminal where leshan client is running
2. Note that the device doesn't show up any more on the `Clients` web page.

Things to try (optional)

1. Locate class `HonoBasedSecurityRegistry` in module `adapters/lwm2m`
2. Take a look at how `SecurityRegistry`'s API is mapped to `HonoClient`

Step 2

Forward Notifications to Hono

What we'll do

In this step we will

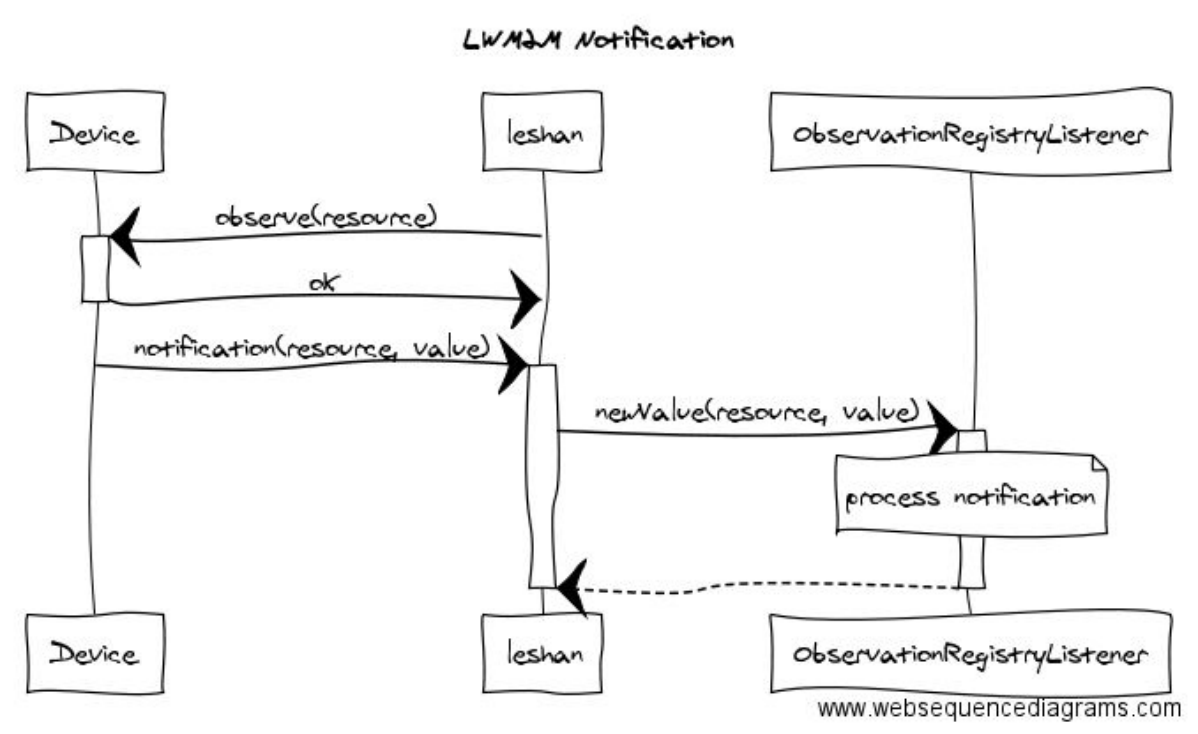
- implement a custom `ObservationRegistryListener` that forwards *notifications* received from LWM2M clients to Hono's Telemetry API
- start up the LWM2M adapter
- start up a consumer for Hono's Telemetry API
- start up a LWM2M client with a specific location
- observe the device's Location object

LWM2M Notification Sequence

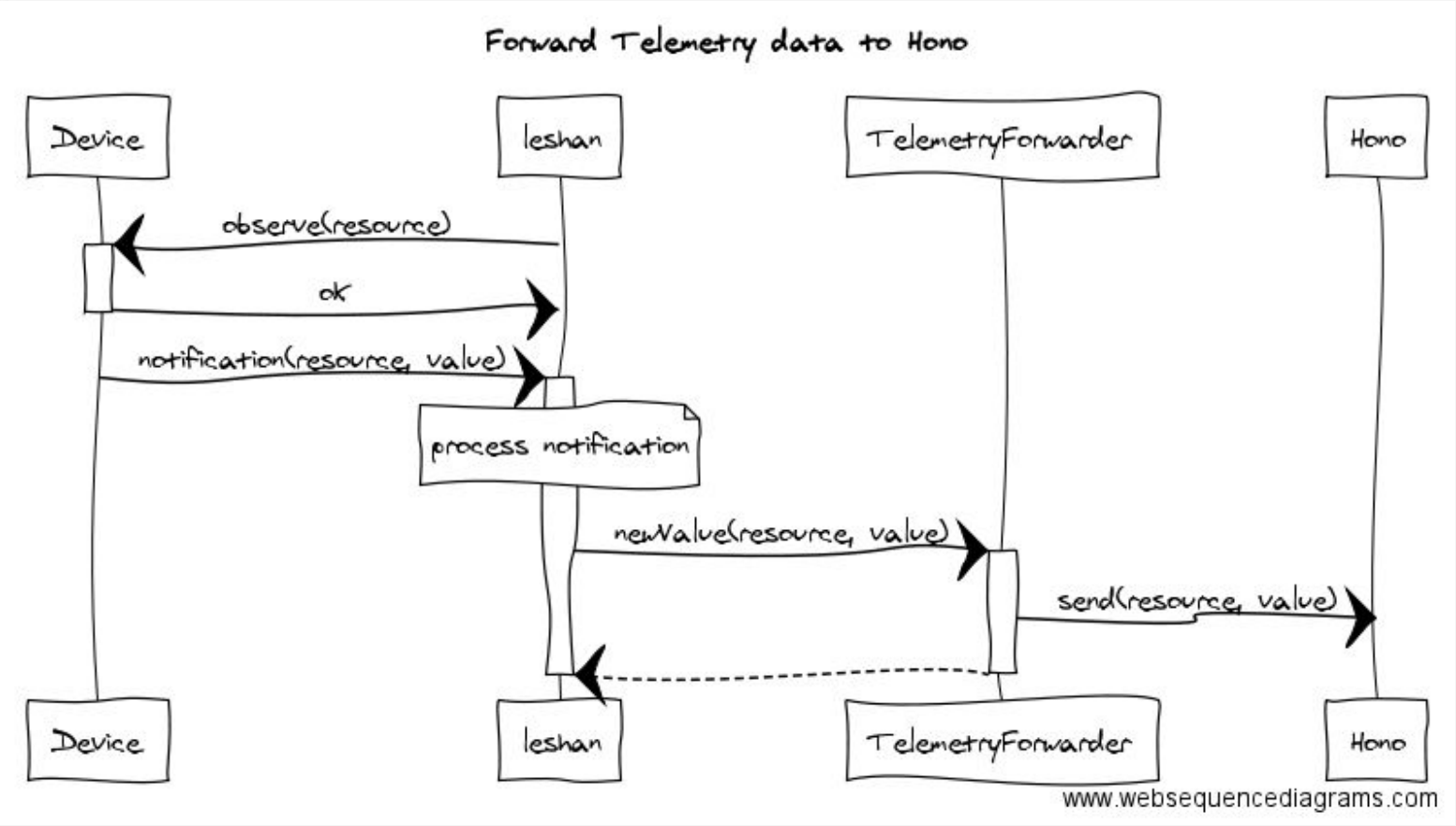
leshan starts observing resource

Device sends a notification for changed resource

leshan forwards the notification to registered listeners



Telemetry Forwarder



Start up LWM2M Adapter

Start up adapter

1. Switch to branch `step2` in the Hono source folder
2. Find class `TelemetryForwarder`
3. Take a look at how `ObservationRegistryListener` 's API is mapped to `HonoClient`
4. Start up LWM2M adapter

```
$> cd $HONO_HOME/adapters/lwm2m
$> mvn spring-boot:run -Dhono.client.host=your_docker_host
```

5. Read through the log and verify that the adapter connects to Hono and uses our custom `TelemetryForwarder` implementation

Start up Telemetry Consumer

Start up adapter

1. Open a new terminal
2. Start up Telemetry consumer

```
$> cd $HONO_HOME/example
$> mvn spring-boot:run -Dhono.client.host=your_docker_host
```

3. Read through the log and verify that the adapter connects to Hono's Telemetry endpoint

Start up Device

Run leshan client from new terminal (analogous to step 1).

This time we add additional parameters to set an initial location (Munich, Theresienwiese)

```
$> java -jar lc.jar -n mydevice -i mypskid -p aabbccdd \
$> -pos 48.131048:11.549892 -sf 0.02
```

Check that device has registered as LWM2M client

1. Open web browser, go to http://localhost:8090/#/clients
2. Read Location object from device to verify position

Start observing Location

1. On device's detail page, click the `Observe` button next to `Instance 0` of the `Location` section
2. Note how current Location is retrieved from device
3. Go to terminal in which leshan client is running and press any one of keys `W`, `A`, `S` or `D` followed by `CR`. This shifts the position north, west, south or east and sends out a notification.
4. Go to terminal where the Telemetry consumer is running and see how telemetry messages are coming in for the updated location.
5. Play around with the observed Location resource

Things to try (optional)

1. Alter the format of the telemetry payload uploaded to Hono

   In order to do so you can modify the `JsonPayloadFactory` class (easy) or create a new implementation of `TelemetryPayloadFactory` (advanced)
2. Add additional meta information about the observed device/object to the telemetry message sent to Hono

   In order to do so you can modify the `TelemetryForwarder` class and add some headers to the telemetry message being sent

Bonus Step

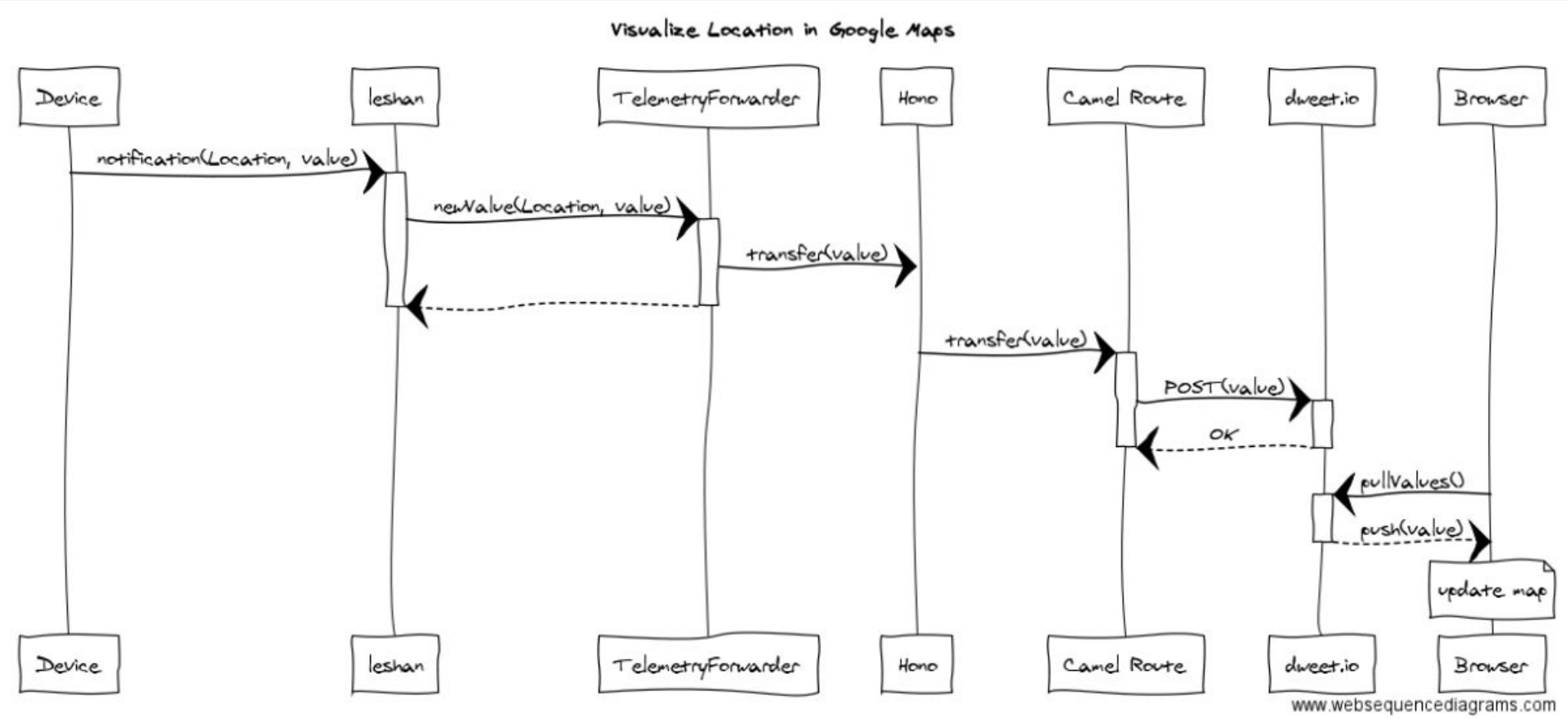
Visualize Location via Google Maps

What we'll do

In this step we will

- set up a dashboard for visualizing our device's location on freeboard.io
- start up the LWM2M adapter
- start up a Camel route forwarding Telemetry data consumed from Hono to dweet.io
- start up a LWM2M client with a specific location
- observe the device's Location and see how it moves on Google Maps :-)

End-to-End Message Flow



Create a Freeboard

1. Go to https://freeboard.io
2. Create an account or log in to your account
3. Go to `My Freeboards` page, enter a name for your board and click `Create New`
4. Click the `ADD` link below `DATASOURCES`
5. Select `Dweet.io` as type and enter a name (no spaces!) for your datasource and a thing name for your device (no spaces!)
6. Click `SAVE`

Add Google Map Widget

1. Click `ADD PANE` on your freeboard
2. Click the wrench symbol on the new pane and increase the columns to 3, click `SAVE`
3. Click the plus symbol on the pane
4. Select `Google Map` as the *type*
5. Click on the `+DATASOURCE` link next to the *latitude* field
6. Select the datasource for your device and click through the hierarchy down to the `latitude` member.
7. Analogously add the path to the longitude value to the *longitude* field.
8. Click `SAVE`

Start up LWM2M Adapter

1. Start up LWM2M adapter

```
$> cd $HONO_HOME/adapters/lwm2m
$> mvn spring-boot:run -Dhono.client.host=your_docker_host
```

2. Read through the log and verify that the adapter connects to Hono

Start up Dweet.io Route

1. Open a new terminal
2. Start up the Dweet.io Camel route

```
$> cd $HONO_HOME/dweet
$> mvn exec:java -Dhono.client.host=your_docker_host \
$> -DthingName=thing_name_from_freeboard
```

   Use the name you used for your thing when you created the dashboard in freeboard.io for the `thingName` parameter

3. Read through the log and verify that the adapter connects to Hono

Start up Device

Run leshan client from new terminal (analogous to step 1).

This time we add additional parameters to set an initial location (Munich, Theresienwiese)

```
$> java -jar lc.jar -n mydevice -i mypskid -p aabbccdd \
$> -pos 48.131048:11.549892 -sf 0.02
```

Check that device has registered as LWM2M client

1. Open web browser, go to http://localhost:8090/#/clients
2. Read Location object from device to verify position

Start observing Location

1. On device's detail page, click the `Observe` button next to `Instance 0` of the `Location` section
2. Go to terminal in which leshan client is running and press any one of keys `W`, `A`, `S` or `D` followed by `CR`. This shifts the position north, west, south or east and sends out a notification.
3. Go to your dashboard in the web browser and observe how the position on the Google Map adjusts to the updated location reported by the device :-)

Things to try (optional)

1. Create additional widgets for data reported by other LWM2M Objects, e.g. Device

   Hint: you will probably need to adjust the Camel route for that purpose and e.g. post data to different thing addresses based on the LWM2M object that the data is reported for (contained in `object` header)

Evaluate the Sessions

Sign in and vote at **eclipsecon.org**

- 1 0 + 1

iot
eclipse.org

Backup

Other Stuff

leshan Client Cheat Sheet

download leshan client

https://drive.google.com/open?id=0B8o-KHF-_cPKTEgiV2UtaDZGaVU

```
$> mv leshan-client-demo-0.1.11-M14-SNAPSHOT-jar-with-dependencies.jar lc.jar
```

display help

```
$> java -jar lc.jar -h
```

run with PSK using initial coordinates (Munich, Theresienwiese)

```
$> java -jar lc.jar -n mydevice -i mypsk -p aabbccdd -pos 48.131048:11.549892 -sf 0.02
```