From Data Exchange to Edge Delivery

A Beginner's Guide for IoT & Edge Developers

Prepared by: Kaleb Gebremichael Gebremeskel (CNIT) on behalf of STF_685 ESTIMED team members

Co-funded by the European Union

Co-funded by EFTA

Outline

This training walks you, step by step, through two standards every edge-IoT developer runs into: **oneM2M**, for how devices and platforms exchange data, and a minimal **ETSI MEC application**, for how that data gets served back close to the user presented here as **two separate, standalone tracks**. No prior knowledge of either required. A **future tutorial** will show how to integrate them (see Part 5, below).

Part 1: oneM2M Basics

- **Connect to a CSE, register an AE, model data with Containers**
- **Discover, group, control access, and get notified.**

Part 2: MEC app Concepts & Access

- **What Is a MEC Application?**
  - Lifecycle, services & the request flow
- **Access & the General Recipe**
  - The 5-step mental model for any MEC app

Part 3: Build & Register (Main Focus)

- **Get Your IDs & Configure**
  - Sandbox ID, App Instance ID, config file
- **Register: Confirm Ready & Subscribe**
  - The three calls every registered app makes

Part 4: Consuming MEC Services

- **Discover & Call a Service**
  - Find the MEC013 Location API, then query it every 5 seconds
- **Two Views, One Lifecycle**
  - Your app's logs alongside the sandbox API Console

Part 5: Integration of oneM2M + MEC (COMING SOON)

ETSI The Standards People

ESTIMED enabling standard IoT & MEC deployments

Part 1: oneM2M Basics

Exchanging Data With a CSE: AE, Container & Subscription

Adapted from the oneM2M Jupyter Notebook Tutorials by Ken Figueredo and Andreas Kraft

Co-funded by the European Union

Co-funded by EFTA

What Does oneM2M CSE Solve and How?

IoT devices speak different languages. The oneM2M CSE acts as a central hub "post office" receiving data from devices, routing messages, providing common IoT services and lets other apps read the information securely and reliably.

IoT Devices
Sensors, actuators, smart devices

→

oneM2M CSE
The "post office" that manages and routes everything

→

Your App
Dashboard, controller, any client

Receives & stores data
From any device, automatically

Alerts & notifications
Tells your app when data changes

Controls who sees what
Access rules for authorised apps

Any protocol
HTTP, MQTT, WebSocket, CoAP

oneM2M *= The international rulebook*

A globally agreed standard for IoT devices like HTTP is for transporting messages across the WWW but designed for sensors and smart devices.

CSE *= The IoT hub / middleware*

Common Service Entity. The server in the middle of your IoT network that receives, stores, and serves device data.

AE (App Entity) *= Any app or device that connects*

Your weather app, dashboard, or Raspberry Pi. All are "AEs" when talking to the CSE.

Resource *= A data object with an address*

Everything stored is a resource like temperature reading, device profile, subscription each with a URL-like address.

Container *= A folder for sensor readings*

Holds a series of data values like a log file for one sensor. Your thermometer pushes in and your app reads out.

Subscription *= A push notification rule*

Tells the CSE: "When new data arrives here, notify me." Saves you from constantly polling for updates.

Connect & Register: CSEBase → AE

The first two calls any app makes: confirm the CSE is alive, then introduce yourself.

NB: *Long attribute names are used here for clarity, oneM2M abbreviates them on the wire as short names to save bandwidth. See the LONG / SHORT tags below.*

Note: *Any HTTP client works here, this tutorial uses curl throughout with no extra setup needed.*

Is the CSE Even Alive?

```
curl -X GET \
  -H 'X-M2M-Origin:CAdmin' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  http://localhost:8081/cse-in

200 OK   (RSC 2000)
LONG  {"m2m:CSEBase": {
  "resourceID": "id-in",
  "resourceName": "cse-in",
  "cseType": 1,
  "supportedReleaseVersions": ["2a","3","4"]
}}
SHORT  {"m2m:cb": {"ri":"id-in","rn":"cse-in","cst":1,
  "srv":["2a","3","4"]}}
# This is the very first call any app makes.
```

Basic Resources to Go deeper

Notebook (Wiki) ↗ | Slides (PDF) ↗ | Video (YouTube) ↗

Can I Get My Own Identity?

```
curl -X POST \
  -H 'X-M2M-Origin:Cmyself' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  -H 'Content-Type:application/json;ty=2' \
  -d '{"m2m:ae":{"rn":"Notebook-AE",
   "api":"NnotebookAE","rr":true,
   "srv":["3"]}}' \
  http://localhost:8081/cse-in

201 CREATED   (RSC 2001)
LONG  {"m2m:ApplicationEntity": {
  "resourceName": "Notebook-AE",
  "App-ID": "NnotebookAE",
  "requestReachability": true,
  "supportedReleaseVersions": ["3"],
  "resourceID": "Cmyself",
  "parentID": "id-in",
  "AE-ID": "Cmyself"
}}
SHORT  {"m2m:ae": {"rn":"Notebook-AE","api":"NnotebookAE",
  "rr":true,"srv":["3"],"ri":"Cmyself",
  "pi":"id-in","aei":"Cmyself"}}
# resourceID = AE-ID = "Cmyself"
```

Model Your Data: Container → Reading

A Container is a folder for one data source and a ContentInstance is the reading inside it.

Where Do I Put My Data?

```
curl -X POST \
  -H 'X-M2M-Origin:Cmyself' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  -H 'Content-Type:application/json;ty=3' \
  -d '{"m2m:cnt":{"rn":"Container"}}' \
  http://localhost:8081/cse-in/Notebook-AE

201 CREATED
 LONG {"m2m:Container": {
  "resourceName": "Container",
  "resourceID": "cnt331511...",
  "currentNrOfInstances": 0,
  "stateTag": 0
}}
201 CREATED
SHORT {"m2m:cnt": {"rn":"Container","ri":"cnt331511...",
  "cni":0,"st":0}}
# No attributes needed, the CSE fills in sensible defaults.
```

How Do I Record One Reading?

```
curl -X POST \
  -H 'X-M2M-Origin:Cmyself' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  -H 'Content-Type:application/json;ty=4' \
  -d '{"m2m:cin":{"cnf":"text/plain:0",
       "con":"Hello, World!"}}' \
  http://localhost:8081/cse-in/Notebook-AE
   /Container

201 CREATED
 LONG {"m2m:ContentInstance": {
  "content": "Hello, World!",
  "resourceID": "cin237148...",
  "stateTag": 1
}}
201 CREATED
 SHORT {"m2m:cin": {"con":"Hello,World!",
  "ri":"cin237148...","st":1}}

# No resource Name given, the CSE invents one which is
read-only after this.
```

Basic Resources to Go deeper

Notebook (Wiki) ↗ | Slides (PDF) ↗ | Video (YouTube) ↗

Model Your Data: Retrieve, Update & Delete

The rest of CRUD. la/ol are virtual shortcuts to the latest/oldest reading.

What's the Latest Reading?

```
curl -H 'X-M2M-Origin:Cmyself' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  http://localhost:8081/cse-in/Notebook-AE
   /Container/la
200 OK
LONG  {"m2m:ContentInstance": {
  "content": "Hello, World!",
  "resourceID": "cin237148...",
  "stateTag": 1 }}
SHORT  {"m2m:cin": {"con":"Hello, World!",
  "ri":"cin237148...","st":1}}
PUT /cse-in/Notebook-AE/Container
curl -X PUT \
  -H 'X-M2M-Origin:Cmyself' -H 'X-M2M-RI:123' \
  -H 'X-M2M-RVI:3' -H 'Content-Type:application/json' \
  -d '{"m2m:cnt":{"mni":10}}' \
  http://localhost:8081/cse-in/Notebook-AE
   /Container

# Only send the attribute you're changing.
```

How Much Has It Collected?

```
curl -H 'X-M2M-Origin:Cmyself' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  http://localhost:8081/cse-in/Notebook-AE
   /Container
200 OK
LONG  {"m2m:Container": {
  "resourceName": "Container",
  "currentNrOfInstances": 1,
  "currentByteSize": 13,
  "stateTag": 1 }}
SHORT  {"m2m:cnt":
{"rn":"Container","cni":1,"cbs":13,"st":1}}
DELETE /cse-in/Notebook-AE/Container
curl -X DELETE \
  -H 'X-M2M-Origin:Cmyself' -H 'X-M2M-RI:123' \
  -H 'X-M2M-RVI:3' \
  http://localhost:8081/cse-in/Notebook-AE
   /Container

# Deletes the Container AND every Content Instance
inside it. Child resources always go with the parent.
```

Basic Resources to Go deeper

Notebook (Wiki) ↗ | Slides (PDF) ↗ | Video (YouTube) ↗

Discover: By Type & By Label

Find resources without knowing their exact names and filter by resourceType or by your own labels.

Which Readings Exist, No Names?

```
curl -H 'X-M2M-Origin:Cmyself' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  'http://localhost:8081/cse-in/Notebook-AE
   /Container?fu=2&ty=4&rcn=6'

# fu=2 conditional, ty=4 ContentInstance,
# rcn=6 = references only
200 OK
LONG  {"m2m:resourceRefList": {
    "resourceRef": [
    {"name": "cin_2LXq5xo5Cy",
     "type": 4,
     "value": ".../Container/cin_2LXq..."},
    # ... 4 more refs
  ]}}
SHORT  {"m2m:rrl": {"nm":"cin_2LXq5xo5Cy","typ":4,
  "val":".../Container/cin_2LXq..."}}
```

Can I Find Just What I Tagged?

```
# First, tag a reading when creating it:
"labels": ["tag:greeting"]

curl -H 'X-M2M-Origin:Cmyself' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  'http://localhost:8081/cse-in/Notebook-AE
   /Container?fu=2&lbl=tag:greeting
   &rcn=8'

# rcn=8 = childResources (full data, not just
references, this time)

200 OK
LONG  {"m2m:Container": {"m2m:ContentInstance": [
  {"resourceName": "CINwithLabel",
   "labels": ["tag:greeting"],
   "content": "Hello, World!"}
]}}
SHORT  {"m2m:cnt": {"m2m:cin": [{"rn":"CINwithLabel",
  "lbl":["tag:greeting"],"con":"Hello, World!"}]}}
```

Basic Resources to Go deeper

Notebook (Wiki) ↗ | Slides (PDF) ↗ | Video (YouTube) ↗

Groups: Create & Fan-Out Retrieve

A Group is a fan-out point and one address that reaches every member at once.

Can I Address Many as One?

```
curl -X POST \
  -H 'X-M2M-Origin:CAdmin' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  -H 'Content-Type:application/json;ty=9' \
  -d '{"m2m:grp":{"rn":"Lights-Group",
   "mid":["cse-in/StreetLight-AE-1/
   Light-Container-1","cse-in/
   StreetLight-AE-2/Light-Container-2"],
   "mnm":10}}' \
  http://localhost:8081/cse-in
```

200 OK
LONG

```
{"m2m:Group": {
"resourceName": "Lights-Group",
"memberIDs": ["cnt65553582761098239 09","cnt208487217618695 4483"],
"resourceID": "grp4814762...",
"currentNrOfMembers": 2
}}
# a Group can mix different resource
```

200 OK
SHORT

```
{"m2m:grp":
{"rn": "Lights-Group",
"mid": ["cnt65553582761098 23909","cnt208487217618695 4483"],
"ri": "grp4814762..",
"cnm":2,
```

Basic Resources to Go deeper

Notebook (Wiki) ↗ | Slides (PDF) ↗ | Video (YouTube) ↗

One Address, Write AND Read?

```
curl -X POST \
  -H 'X-M2M-Origin:Cmyself' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  -H 'Content-Type:application/json;ty=4' \
  -d '{"m2m:cin":{"cnf":"text/plain:0",
       "con":"lights-on"}}' \
  http://localhost:8081/cse-in
   /Lights-Group/fopt

# aggregated Response, 2× 201 Created, one
ContentInstance per container


curl -H 'X-M2M-Origin:Cmyself' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  http://localhost:8081/cse-in
   /Lights-Group/fopt/la

# aggregated Response, 2× 200 OK, the latest reading
from each
member's Container, in one request.
```

Access Control: Default Deny, Then Grant

By default only a resource's creator can touch it. Here's the real 403, then the fix.

Why Was I Just Rejected?

```
# CstreetLight-AE-1 tries to POST into
# StreetLight-AE-2's container:
curl -X POST \
  -H 'X-M2M-Origin:CstreetLight-AE-1' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  -H 'Content-Type:application/json;ty=4' \
  -d '{"m2m:cin":{"cnf":"text/plain:0",
       "con":"lights-on"}}' \
  http://localhost:8081/cse-in
   /StreetLight-AE-2/Light-Container-2

403 FORBIDDEN  (RSC 4103)
{"m2m:debugInfo":
  "originator has no privileges
   for CREATE"}

# Only CstreetLight-AE-2, the creator, can access its
own resources by default.
```

How Do I Grant Access?

```
# Request Body
LONG  {"m2m:AccessControlPolicy": {
 "resourceName": "Streetlight-ACP-2",
 "privileges": {"accessControlRule": [
  {"accessControlOriginators":
     ["CstreetLight-AE-1"],
   "accessControlOperations": 1},
  {"accessControlOriginators":
     ["CstreetLight-AE-2"],
   "accessControlOperations": 63}]},
 "selfPrivileges": {"accessControlRule":
  [{"accessControlOriginators":
     ["CstreetLight-AE-2"],
    "accessControlOperations": 63}]}
}}
SHORT  {"m2m:acp": {"rn":"Streetlight-ACP-2",
  "pv":{"acr":[{"acor":["CstreetLight-AE-1"],"acop":1},
   {"acor":["...AE-2"],"acop":63}]},
  "pvs":{"acr":[{"acor":["CstreetLight-AE-2"],"acop":63}]}}}

curl -X POST \
  -H 'X-M2M-Origin:CstreetLight-AE-2' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  -H 'Content-Type:application/json;ty=1' \
  -d '{...body as shown above...}' \
  http://localhost:8081/cse-in
   /StreetLight-AE-2.
```

Basic Resources to Go deeper

Notebook (Wiki) ↗ | Slides (PDF) ↗ | Video (YouTube) ↗

Access Control: Assign the Policy & Verify

An ACP does nothing until you attach it. Then the exact same request that failed now succeeds.

How Do I Attach the Policy?

```
PUT /cse-in/StreetLight-AE-2/
  Light-Container-2 HTTP/1.1
X-M2M-Origin: CstreetLight-AE-2
 # Request Body
LONG {"m2m:Container": {
  "accessControlPolicyIDs": [
    "cse-in/StreetLight-AE-2/
     Streetlight-ACP-2" ]
}}

curl -X PUT \
  -H 'X-M2M-Origin:CstreetLight-AE-2' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  -H 'Content-Type:application/json' \
  -d '{"m2m:cnt":{"acpi":[
   "cse-in/StreetLight-AE-2/
    Streetlight-ACP-2"]}}' \
  http://localhost:8081/cse-in
   /StreetLight-AE-2/Light-Container-2

200 OK  (Updated)
"accessControlPolicyIDs":
  ["acp1132200..."]  # unstructured now
# A resource can list more than one ACP and matching just one is
enough.
```

Did the Permission Take Effect?

```
# Same request that got 403 earlier:
POST /cse-in/StreetLight-AE-2/
  Light-Container-2
  X-M2M-Origin: CstreetLight-AE-1
  Content-Type: application/json;ty=4
 # Request Body
  LONG {"m2m:ContentInstance": {
  "contentInfo": "text/plain:0",
  "content": "lights-on"}}

curl -X POST \
  -H 'X-M2M-Origin:CstreetLight-AE-1' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  -H 'Content-Type:application/json;ty=4' \
  -d '{"m2m:cin":{"cnf":"text/plain:0",
       "con":"lights-on"}}' \
  http://localhost:8081/cse-in
   /StreetLight-AE-2/Light-Container-2

200 OK (LONG)
{"m2m:ContentInstance": {
  "content": "lights-on",
  "resourceID": "cin1362971...",
  "parentID": "cnt7630803..."}}
200 OK (SHORT)
"m2m:cin": {"cnf": "text/plain:0", "con": "lights-on", "ri": " cin136
2971…","pi": " cnt7630803…",}}
```

Basic Resources to Go deeper

Notebook (Wiki) ↗ | Slides (PDF) ↗ | Video (YouTube) ↗

Notifications: Subscribe & Receive

Register once, then the CSE pushes to you the instant a matching resource event happens.

Can the CSE Notify Me?

```
POST /cse-in/StreetLight-AE-2/
  Light-Container-2 HTTP/1.1
Content-Type: application/json;ty=23
# Request Body
 LONG {"m2m:Subscription": {
  "resourceName": "Subscription",
  "notificationURI": [notificationURL],
  "notificationContentType": 1,
  "eventNotificationCriteria": {
    "notificationEventType": [1,2,3,4]}
}}
```

```
SHORT {"m2m:sub":
{"rn":"Subscription",
"nu":[notifURL],
"nct":1,
"enc":{
"net":[1,2,3,4]}}}
```

```
curl -X POST \
  -H 'X-M2M-Origin:CstreetLight-AE-1' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  -H 'Content-Type:application/json;ty=23' \
  -d '{...body as shown above...}' \
  http://localhost:8081/cse-in
   /StreetLight-AE-2/Light-Container-2
201 CREATED
# The CSE first sends a verification
# request to notificationURI to check
# it's reachable, before confirming.
```

Basic Resources to Go deeper

Notebook (Wiki) ↗ | Slides (PDF) ↗ | Video (YouTube) ↗

What Does That Look Like?

```
POST /cse-in/StreetLight-AE-2/
  Light-Container-2
X-M2M-Origin: CstreetLight-AE-1

 LONG {"m2m:ContentInstance": {
  "contentInfo": "text/plain:0",
  "content": "lights on"}}

curl -X POST \
  -H 'X-M2M-Origin:CstreetLight-AE-1' \
  -H 'X-M2M-RI:123' -H 'X-M2M-RVI:3' \
  -H 'Content-Type:application/json;ty=4' \
  -d '{"m2m:cin":{"cnf":"text/plain:0",
       "con":"lights on"}}' \
  http://localhost:8081/cse-in
   /StreetLight-AE-2/Light-Container-2

201 CREATED — as usual.

# But now your notification server ALSO receives a POST
containing this same Content Instance automatically.
# No polling, every, event types 1–4 (update / delete /
child-created / child-deleted.)
```

Try every request and response Yourself, One Click Away

Try every request and response body straight from the sandbox. Here's where the official series takes you next.

Every example here ran against a real local CSE (ACME) inside the official Jupyter notebooks. The same environment is one click away, no install required.

Zero Install

Click the Binder link, wait about a minute, then open __START__.ipynb. A full CSE (ACME) runs inside the notebook.

Run instantly on Binder

Useful links:

- **All 8 episodes, videos + slides**: wiki.onem2m.org
- **Reference CSE used throughout:**
  - acmecse.net
  - github.com/ankraft/ACME-oneM2M-CSE
  - recipes.onem2m.org

Local Setup

```
$ git clone < the repo>
$ cd onem2m-jupyter-notebooks

$ Create a virtual environment
$ pip3 install -r requirements.txt
$ jupyter lab --NotebookApp.token='' __START__.ipynb
```

github.com/oneM2M/onem2m-jupyter-notebooks

ETSI The Standards People

ESTIMED enabling standard IoT & MEC deployments

Part 2: How to Connect Your Own App?

Concepts, Lifecycle & Sandbox Access for Edge-IoT Developers

Adapted from ETSI MEC Sandbox resources and MEC Sandbox Help WiKi

Co-funded by the European Union

Co-funded by EFTA

Multi-Access Edge Computing

Run your app beside the base station, not in a distant cloud.

ETSI MEC moves compute to the edge of the mobile network (next to 4G/5G towers or base stations). Your app gets real-time location & network data responding in milliseconds, not seconds.

Ultra-low latency
ms, not seconds

Location-aware
Real-time device tracking

Network-aware
Signal, load, handovers

Open REST APIs
Free sandbox to test

What Problem Does a MEC Application Solve?

A MEC application solves a **latency problem**: instead of running your business logic far away in a cloud data center, it runs **on the MEC platform itself,** physically close to the radio network and the user. The ETSI MEC platform acts as a **broker sitting next to the base station**: your app **registers to it**, **discovers MEC services** like Location, and **consumes them over plain HTTP**.

What your app can ask the Edge?

ETSI MEC exposes these RESTful APIs , call them with HTTP, no special SDK required.

"Where are my users?"

Location API: Track user positions in real time, detect zone entry/exit, and trigger location-based actions in your app.

```
GET /location/v2/users?zoneId={id}
```

"How is my users' connection quality?"

Radio Network Info: Read signal strength, cell load, and UE throughput to adapt your app behavior, for example reduce video quality on a congested cell.

```
GET /rni/v2/queries/rab_info
```

"Is my user on Wi-Fi or mobile?"

WLAN Information: Detect Wi-Fi vs 4G transitions and optimize handovers, useful for video streaming or real-time apps.

```
GET /wai/v2/queries/sta/sta_information
```

"Can this edge host run my app?"

App Enablement API: Register your MEC app with the MEC platform and discover other MEC services running on the same edge host. **A MEC app** is any application that runs HTTP requests close to IoT devices at the network edge (instead of far away in the cloud)

```
POST /mec_app_support/v2/registrations
```

"How do I keep my app close to a moving user?"

App Mobility : Detect when a user hands over to a new base station and migrate your app to the nearest MEC host automatically.

```
POST /appMobilityService
```

Building Any MEC Application: The General Recipe

Every MEC app needs a platform to run on

A MEC platform is compute that sits right next to the mobile network, instead of far away in the cloud. You can start on a free public sandbox, or run your own private one.

Every MEC app needs an identity

Your App Instance ID

The platform hands your MEC app a unique ID the moment you register it. Every message your app sends carries this ID, think of it like a username for your app

How You Talk to It

Plain HTTP requests: GET to fetch data, POST to register or send updates
No special SDK needed, any language works(GO, python,..)

Every MEC App Follows These 5 Steps

1 Access and Run

Sign up for a MEC sandbox and your own personal test environment(Sandbox) is created for you automatically.

Usually free to try

2 Register your app

Create an application instance in the platform's dashboard. It hands back a unique ID that identifies your app from now on.

Your app's ID card

3 Say "I'm ready"

When your app starts up, it tells the platform it booted successfully and asks to be warned before it's ever shut down.

The "confirm ready" handshake

4 Find what you need

The platform exposes a catalog of ready-made services like location, network quality, and more. Your app looks through it for the one it needs.

Called "service discovery"

5 Use it, then let go

Call the service to do your job **(e.g. track a device's location)**, then unregister everything cleanly when your app is done.

Register → Say you're ready → Find a service → Use it → Shut down cleanly

MEC Application: The Ground Rules

```
# Any HTTP-capable client works: curl, Postman, your
own app
curl -X GET "{basePath}/mec_service_mgmt/v1/services"
\
  -H "accept: application/json"
// {basePath} = the link from the Try-it area, scoped
to YOUR sandbox
```

Rule	What It Means
Base path	Copy it from the Try-it area, it routes requests to your specific user sandbox
Appending paths	Append the desired MEC services endpoint path, plus query and body parameters when necessary
Any client works	Any external application or process with HTTP client capabilities, can be used to send requests
Full endpoint details	Access the Swagger UI client for the supported endpoint paths and parameters
Notification URLs	Must be publicly reachable, or the API Console shows a 500 Internal error
Application Instance IDs	Must be provisioned before use manually here, since the sandbox emulates orchestration

Why Is ID Provisioning Manual Here?

In a Real Deployment

The Application Instance ID is provisioned automatically, at orchestration time, as part of deploying your App. No manual steps the orchestrator handles it as your app is placed on an edge host.

`In This Sandbox : Application Instance IDs must therefore be provisioned manually.`

Notification Reachability, in Practice

Reachable callback notifyURL: `http://my.callback.com/... (public)`
Result: `200 OK` API Console shows the request & response data

Unreachable callback notifyURL: `http://localhost:8080/... (private)`
Result: `500 Internal error` Shown in the API Console — the request never got there

ETSI The Standards People

ESTIMED enabling standard IoT & MEC deployments

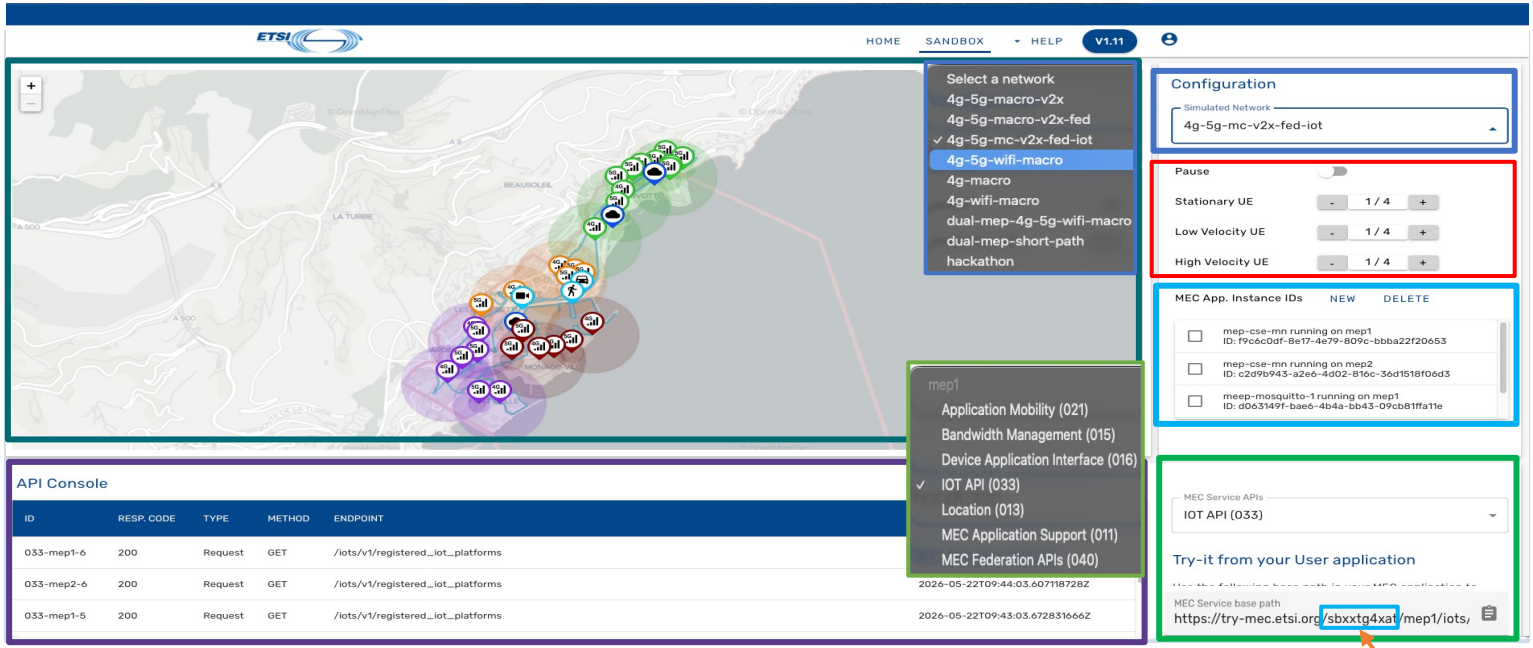
Part 3: Build & Register Your MEC App

The Complete Lifecycle: From Sign-Up to a Registered, Running App (MEC011) with Go

Co-funded by the European Union

Co-funded by EFTA

Where Do You Even Start?



Steps 1–3 of 5: Registering Your Application

1 Go to the sandbox

Sign in to the ETSI MEC Sandbox (or your organization's MEC platform). **Once you sign in**, you are provisioned with your **personal sandbox that has six panel**s to build and test MEC applications.

2 Create an Application Instance

In the app instance manager panel, create a new application. The platform generates a unique Application Instance ID for it. This is your app's identity from now on.

3 Obtain the Sandbox ID from Try it Area

Your personal sandbox also has its own ID (e.g. sbxxtg4xat). You'll need it together with the App Instance ID for every request.

4 Configure your app

Open your app's configuration file and replace the placeholder values with the real Application Instance ID and Sandbox ID from Steps 2 & 3.

5 Build the client & server

Your app needs two things: an HTTP client **to call the platform**, and an HTTP server so the platform can call you back. Any language works, this tutorial uses Go. *With your IDs in the config file and your client/server code in place, your app is ready to officially register itself, that's next Steps.*

Step 4: Replace the Place holders in your config file With Your Real IDs

Every MEC app reads its settings from one config file.

① Config file
Ships with placeholder values (a sample appid, sandboxName) copied from the ETSI example. Safe to read, not to run.

② Your edits (before first run)
Set sandboxName and appid to the values you generated in the sandbox UI, and ueAddress to the UE you want to track.

③ entrypoint.sh does this for AdvantEDGE
When mode is advantedge, entrypoint.sh regenerates the file from MEEP_* environment variables at container start, no manual editing.

The three fields you will always edit:

Setting	What it controls	Example value
appid	Your app instance ID from the sandbox UI	88043521-1577-4ebe-bd92-70f87d0e5987
sandboxName	Identifies your personal MEC Sandbox instance	sbxykqjr17
ueAddress	The UE (car) whose location demo1 tracks	10.100.0.1

Every field is documented as a comment directly inside config file

Step 5: Build A Client to Call Out, a Server to Listen in Go

```
Dir Structure

app/
├── main.go
├── routers.go
├── app_instance.yaml
├── Dockerfile / entrypoint.sh
├── docker_build.sh / docker_run.sh
└──
```

File	What it does
main.go	The HTTP client: loads config, calls the platform, tracks a service, shuts down
routers.go	The HTTP server: a tiny REST API so the platform can reach your app back
app_instance.yaml	Every Config setting we need: sandbox, app ID, ports, callback URL
Dockerfile	Base golang image + entrypoint.sh as the container entrypoint
entrypoint.sh	Builds app_instance.yaml from MEEP_* env vars, then execs app
docker_build.sh / docker_run.sh	Compile inside a golang container, then run app with your config

Base path:

use the provided link as a base path and append the desired service endpoint path.
Eg In Go: cfg.BasePath = mecUrl+"/sandbox-api/v1", then each call appends its own path.

Your callback URL must be reachable

if it isn't publicly reachable, "the request will fail and the API console will display a 500 Internal error." That's exactly where routers.go comes in.

```
Eg : Base path + append the path for confirm ready request : curl -X POST {basePath}/mec_app_support/v2/applications/{id}/confirm_ready
```

Helper Scripts

build-demo1.sh Install: `docker pull golang && ./docker_build.sh`

docker_run.sh Install: `docker run --publish 8080:8080 -v$PWD:/opt/local/etsi/app golang ...`

One Function Every MEC Call Reuses

```
func mecRequest(method, path string,
  body []byte) (*http.Response, error) {

  req, _ := http.NewRequest(method,
    basePath+path, bytes.NewReader(body))
  req.Header.Set("Content-Type",
    "application/json")
  return http.DefaultClient.Do(req)
}
```

Argument	What You Pass
method	"GET", "POST", or "DELETE" — same as any REST call
path	Just the part after basePath, e.g. "/applications/{id}/confirm_ready"
body	JSON payload as bytes — nil for GET and DELETE
Returns	The raw *http.Response — decode resp.Body yourself

Then, Every MEC Step Is Just a Different Call

Set basePath Once

```
basePath = mecUrl + "/" + sandboxName + "/" + mecPlatform
+ "/sandbox-api/v1"
```

Reuse It for Every Step

```
mecRequest("POST",  ".../confirm_ready", body)    // confirm ready
mecRequest("POST",  ".../subscriptions", body)     // subscribe
mecRequest("GET",   ".../services", nil)           // discover
mecRequest("GET",   ".../queries/users?...", nil)  // query location
mecRequest("DELETE",".../subscriptions/{id}", nil)  // clean up
```

No generated client, no external SDK, just net/http and encoding/json from Go’s standard library.

Two Small Types You’ll Decode Every Response Into

```
type Service struct {
  Name  string `json:"serName"`
}
```

```
type Location struct {
  Latitude, Longitude float64
}
```

Send the Confirm Ready Request

Confirm Ready Is a Handshake, Not Just a Message

```
POST .../applications/{appId}/confirm_ready
{ "indication": "READY" }
... up to 10 seconds later ...
POST .../applications/{appId}/confirm_termination
{ "operationAction": "TERMINATING" }
// The Go version, using mecRequest:
func confirmReady(appID string) error {
  body := []byte(`{"indication":"READY"}`)
  _, err := mecRequest("POST",
    "/applications/"+appID+"/confirm_ready", body)
  return err
}
```

Start up: Two Calls

Step 1: You speak first
Your app always initiates: it says "I'm ready" before the platform will do anything else with it. If this call fails, there's no point continuing, the app exits.

Question	Answer
Why send it?	It tells the platform your app started up correctly and is ready to operate
Who do you send it to?	The platform's Application Lifecycle Management API (MEC011)
What builds the URL?	Your Sandbox ID + App Instance ID from the config file, from Steps 2-4
When do you send it?	Immediately, as the very first thing your app does after it starts
What comes back?	204 No Content on success, no error means you're registered
Where do you send the notification?	A callback reference, where notifications will be sent, required in every subscription request

Step 2: Then subscribe for termination notification

Right away, your app also asks to be warned before it's ever shut down. Only needed if you want a graceful shutdown, skip it and the platform just kills the app.

Subscribe to Termination Notifications

```
POST /.../applications/{appId}/subscriptions
Body: {
  "callbackReference": "https://your-app/termination",
  "appInstanceId": "{appId}" }
Response: 201 Created + a subscription ID
```

In the code	What it does
Why subscribe?	So the platform can warn your app before it kills and cleans up
subscribeToTermination()	Called automatically at the end of confirm_ready
AppTerminationNotificationSubscription	JSON struct: SubscriptionType, CallbackReference, AppInstanceId
CallbackReference	= yourCallbackURL + "/termination" where the platform POSTs
parseSubscriptionID()	Parses that header down to just the subscription ID
Why keep that ID?	You need it later to update or delete this exact subscription

Every Subscription Needs Two Things

1. Notification URL

Every subscription needs a notification URL where notifications will be sent.

2. Filter Criteria

Filter criteria specify which events to register for.

The Whole Step, as One Go Function

```
func subscribeToTermination(id string) (string, error) {
  b, _ := json.Marshal(map[string]string{"subscriptionType": subType,
    "callbackReference": cbURL + "/termination", "appInstanceId": id})
  resp, err := mecRequest("POST", "/applications/"+id+"/subscriptions", b)
  if err != nil { return "", err }
  return parseSubscriptionID(resp.Header.Get("Location")), nil
}
```

Heads up, wrapping varies: Location Service returns resourceURL as a plain string. While RNIS/WAIS/AMS/Mp1 nest it under _links.self.href instead, check the Swagger UI per MEC API.

The Server to Serve a Callback From Step 5, in Code

```
type HttpRoute struct {
  Name, Method, Pattern string
  HandlerFunc http.HandlerFunc
}
var routes = HttpRoutes{
  {"Index","GET","/", Index},
  {"Callback","POST",
   "/termination", handleCallback},
}
```

In the code	What it does
NewRouter()	Loops the routes slice and registers each one on a mux.Router
handleCallback(w, r)	Logs the incoming request, replies 200 OK
handlers.CORS(methods, header)	Wraps the router, allowing OPTIONS/DELETE/GET/HEAD/POST/PUT
http.ListenAndServe(...)	Starts the server on config.CallbackPort, inside its own goroutine

Two Callback URLs: A Health Check and a Warning Bell

URL 1: "Are you alive?"

A simple GET request that anyone can send to check the app is up and responding. It just replies "Hello World!" proof the server is listening.

URL 2: "I'm about to shut you down"

This is the platform's way of keeping its promise : `it sends a heads-up before terminating the app`

```
func handleCallback(w http.ResponseWriter, r *http.Request) {
  log.Println("platform callback:", r.URL.Path)
  w.WriteHeader(http.StatusOK)
}
```

Two Small Building Blocks Behind the Scenes

Routing Import: `"github.com/gorilla/mux"`
Gives: `mux.NewRouter().StrictSlash(true)` Pattern-based path matching

CORS Import: `"github.com/gorilla/handlers"`
Gives: `handlers.CORS(methods, header)(router)` Wraps NewRouter() before it's served

ETSI The Standards People

ESTIMED enabling standard IoT & MEC deployments

Part 4: Consuming MEC Services

Once Registered, Here's How Your App Uses the Platform's Services (e.g. MEC013 Location)

Co-funded by the European Union

Co-funded by EFTA

The Complete Picture, Before We Consume a Service

You've built and registered the app. Here's where discovering and consuming a service fits in.

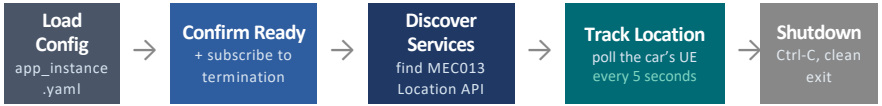
A MEC application, it runs on the MEC platform itself and physically close to the radio network and the user. Once registered, it consumes MEC platform services (like Location) to deliver business logic closer to the user than a distant cloud app ever could.

Config-driven
YAML file, no hardcoding

Lifecycle-aware
confirm_ready & termination

Service-aware
Finds the MEC013 Location API

REST callback server
Receives termination notices



Lead By: STF 685 ESTIMED

Useful links:

- *demo1 source & README (GitHub)*
- *ETSI GS MEC 011 & MEC 013 specifications*

What we did, step by step

"Did my config load correctly?"

Confirm Ready: Tells the MEC platform the app booted successfully and is ready to run, as defined in ETSI GS MEC 011 Clause 5.2.2.

```
POST /mec_app_support/v2/applications/{id}/confirm_ready
```

"Is the platform ready for me?"

Subscribe to Termination: Registers a callback URL so the platform can warn demo1 before terminating the application instance.

```
POST /mec_app_support/v2/applications/{id}/subscriptions
```

"Which services can I use?"

Discover Services: Lists every MEC service the platform exposes; demo1 scans the list for the MEC013 Location API.

```
GET /mec_service_mgmt/v1/services
```

"Where's the car right now?"

Query Location: Fetches the live latitude and longitude of the tracked UE (car) **from the MEC013** Location API every 5 seconds and prints latitude/longitude to the console.

```
GET /location/v3/queries/users?address={ueAddress}
```

"Did I clean up on exit?"

Graceful Shutdown: On Ctrl-C, demo1 deletes its subscriptions and frees resources before exiting cleanly (ETSI GS MEC 011 Clause 5.2.6b).

```
DELETE /mec_app_support/v2/.../subscriptions/{id}
```

Discover and Use a Service

```
services, err := getAvailableServices()
// → GET .../mec_service_mgmt/v1/services
for _, s := range services {
  if strings.Contains(s.Name, "mec013") {
    locationSvc = s
}}
// POST .../subscriptions
{"subscriptionType":"SerAvailabilityNotificationSubscription",
"filterCriteria":{"serNames":["mec013"],"states":[...]}}
loc, err := getLocation(targetAddress)
// → GET .../location/v3/queries/users?address=...
```

In the code	What it does
getAvailableServices()	GETs the platform's full service list and decodes it into []Service
strings.Contains(s.Name, "mec013")	Scans the list for the Location API, exits if it isn't found
getLocation(address)	GETs /location/v3/queries/users?address={ue Address}
LocationResponse	The decoded JSON response holds Latitude[] and Longitude[]

First Find the Right Service, Then Use It Repeatedly

① Find MEC013 (Location)

Location Service gives geospatial and network location information, user/zone location plus tracking & zonal-status events.
If none matches "mec013", logs it and returns.

② Start the tracking goroutine

`main()` launches a goroutine that runs the loop below for as long as run stays true: `go func(){...}()`

```
func getAvailableServices() ([]Service, error) {
  resp, err := mecRequest("GET", "/mec_service_mgmt/v1/services", nil)
  if err != nil { return nil, err }
  var s []Service; return s, json.NewDecoder(resp.Body).Decode(&s)
}
func getLocation(addr string) (*Location, error) {
  resp, _ := mecRequest("GET", "/location/v3/queries/users?address="+addr, nil)
  var loc Location; return &loc, json.NewDecoder(resp.Body).Decode(&loc)
}
```

Two Views of Every Interaction: Your Logs vs the API Console

Everything your app sends also shows up live in the sandbox's API Console , giving you both the client and the platform perspective on the same event.

① Confirm Ready
Your app logs a 204 response, the console shows a new POST .../confirm_ready request against your App Instance ID.

② Termination Subscription
Your app logs the new subscription ID, the console lists it under registered subscriptions, with your callback URL attached.

③ Location Service Request
Your app logs the coordinates it received, the console shows the GET request to the Location Service and its JSON response, side by side.

Side by side, across the whole lifecycle:

Event	Your App's Log	What the API Console Shows
Confirm Ready	"confirm ready: 204 No Content"	New request: POST confirm_ready → 204
Termination Subscription	"subscribe termination: 201 Created"	New entry under your app's subscriptions
Location Request	"latitude = ... longitude = ..."	GET .../location/v3/... → 200 OK + JSON body

This is a great debugging habit: if your app logs an error, check the API Console to see whether the request even arrived.

Recap & Next Steps

You have now walked through building and registering a complete MEC application in Go: sign-up, IDs, config, confirm_ready, subscription, graceful termination and the skeleton every MEC app shares. The **same pattern** applies once you start consuming services too, Location was just one example. Swap the endpoint, keep the same registration lifecycle.

What You Learned

- **Exchange Data with oneM2M**
  - Connect, register, model data, discover, group, secure, notify
- **Build & Register a MEC App**
  - Sandbox login → IDs → config → confirm_ready → subscribe → terminate
- **Two Views: Logs & API Console**
  - Every event visible from both the client & platform side
- **Consume a Service (Secondary Focus)**
  - Discover services, then call the Location Service

Where To Go Next

- **Explore More MEC APIs**
  - **Radio Network Info, WLAN Info, MEC 033, MEC 046**
- **PART 5 (COMING SOON): One Workflow, Two Standards**
  - **Extend our App to Discover oneM2M CSE running in the MEC sandbox**
    - **MEC 033 lists your oneM2M platforms**
    - **MEC 046 their resources, then create an AE, Container & Content Instance, just like Part 1**

Ready to Build? Join the ESTIMED Hackathon

This tutorial gets you ready, here's how to connect with the community and register for the hackathon.

Register Here

Build a scalable, cross-industry solution bridging IoT and Edge. Open from July 1 through September 1

- **Registration Link**

New to This? Start Here

Haven't set up your workspace yet? This tutorial walks you through the oneM2M CSE and MEC Sandbox setup, no prior experience needed.

- Get Your Workspace Ready, how to run the MEC Sandbox and a oneM2M CSE
- EDGE-IoT Systems, Use Cases & Reference Architecture Foundations

Visit the ESTIMED Website

For more information about the ESTIMED project and hackathon details, visit the official website:
ESTIMED Website: https://estimed.etsi.org/

Co-funded by EFTA

Co-funded by the European Union