Platform Access & Environment Setup to Get you Started

A Beginner's Guide for IoT & Edge Developers

Prepared by: Kaleb Gebremichael Gebremeskel (CNIT)

Co-funded by the European Union

Co-funded by EFTA

Outline

This guide walks you through the two building blocks you need to participate in the hackathon: **MEC** moves your app closer to the user giving you real-time location and network data at the edge. The **oneM2M CSE** connects and manages your IoT devices through a single hub. Together they form the platform for building smart, real-time EDGE-IoT applications.

Part 1 — ETSI MEC

- **Overview & Service APIs**
  - Location, Radio, WLAN, App APIs
- **Configuring ETSI MEC**
  - Cloud sandbox & local install options
- **MEC Sandbox Dashboard**
  - Map, UEs, APIs and console panels

Part 2 — oneM2M CSE

- **Architecture & Key Concepts**
  - CSE, AE, Resources, Containers
- **Two Open-Source CSE Options**
  - **ACME CSE**
    - Overview, install, and configuration
  - **tinyIoT CSE**
    - Overview, install, and configuration

ETSI The Standards People

ESTIMED enabling standard IoT & MEC deployments

Part 1:ETSI MEC

Multi-Access Edge Computing: Overview, APIs & Configuration

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

Lead By: ETSI ISG MEC
Useful links:

- *ETSI MEC Sandbox user interface guide*
- *ETSI MEC Sandbox Deployment*

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
```

ETSI MEC Installation in two ways

Option A: Cloud sandbox (recommended for beginners)

mec-platform.etsi.org free browser-based sandbox, no install, no hardware required. Create a free ETSI account and Sign in with ETSI account or OAuth.

Option B: Local Installation Options (recommended when you need local deployment)

Manual Installation

Step-by-step guide: labs.etsi.org/rep/mec/etsi-mec-sandbox-frontend
Build & Deploy MEC Sandbox

Automated Installation (Ansible)

Ansible playbook: labs.etsi.org/rep/mec/etsi-mec-sandbox (playbooks/README)
Use when no public sandbox required.

Access & Run in 5 steps

1 Sign in

Go to mec-platform.etsi.org → sign in (ETSI account or OAuth). Your personal sandbox provisions automatically.

Free

2 Pick a network scenario

Choose: 4g-5g-wifi-macro (general), 4g-5g-mc-v2x-fed-iot with dual-mep (mobility). Deploys on a Monaco city map.

Start with 4g-5g-wifi-macro

3 Add terminal devices (UEs)

Add pedestrians, vehicles or objects. Set velocity (stationary / low / high).

UEs attach to PoAs automatically

4 Call a MEC API — Swagger UI

In Try-it area: select an MEC service API from drop down → Swagger UI opens with base URL pre-filled → explore the supported endpoints and required parameters for the selected MEC service → click Execute to send requests → view live JSON responses directly in the browser.

No code needed in the browser

5 Connect your own app

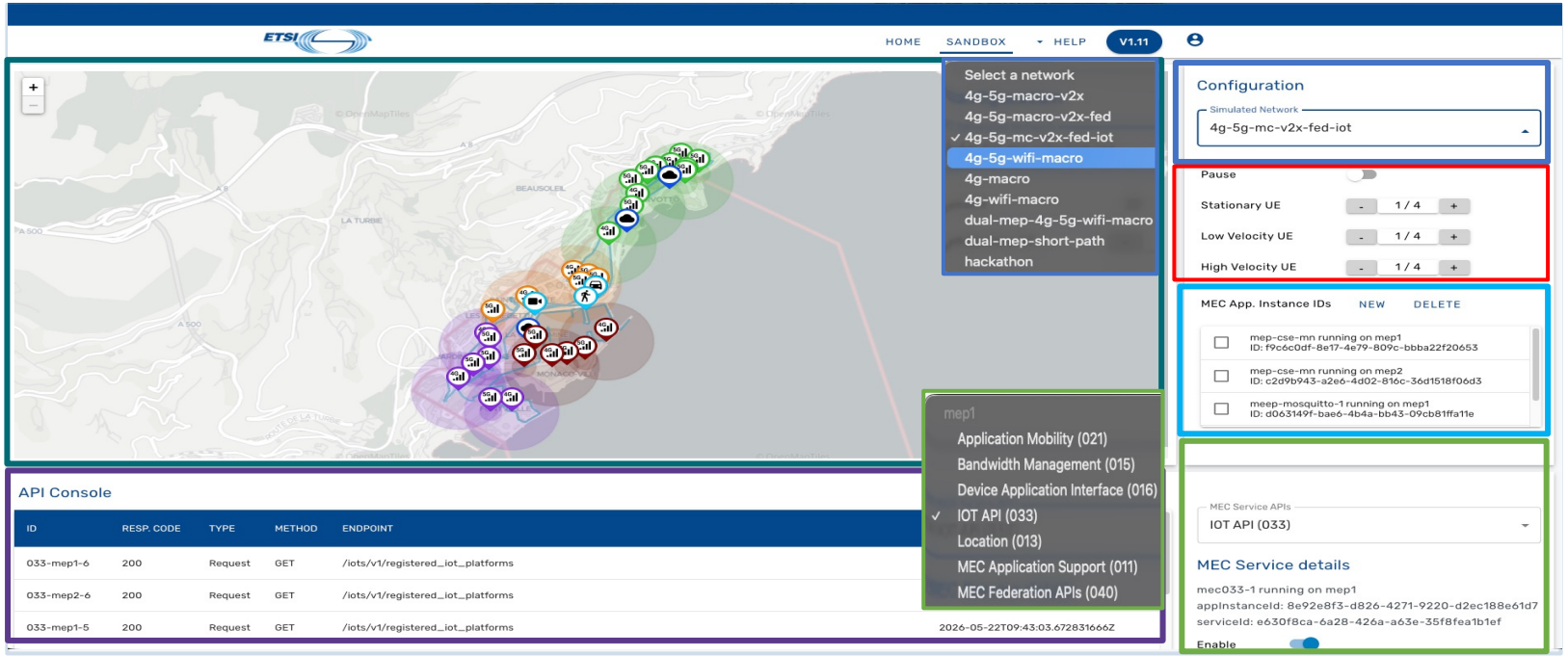
Copy the base URL **(https://mec-platform.etsi.org/{sandboxid}/selectedMEC services API endpoint)** from Try-it area dashboard. Use it in any application that can send http requests or in curl, Postman by appending the endpoint path and parameters of specific MEC service.

For example this request calls the location API for user location and returns a result as a JSON response.

```
curl -X GET "{baseURL}/location/v2/users" -H "accept: application/json"
```

ETSI MEC Sandbox Dashboard Explained

Once you sign in, you are provisioned with your **personal sandbox that has six panel**s to build and test MEC applications. Each **MEC application** is created with a **unique Application Instance ID**, which acts as its identity within the MEC sandbox and is used when accessing certain MEC services. **Each panel** controls a different aspect of your workflow, from placing virtual devices on a map to creating MEC app and calling live MEC APIs via Swagger. **The descriptions below explain what each panel does.**



Map Panel

Configuration Panel

UE Panel

MEC app instance manager Panel

Try it area

API console

Map Panel: Monaco city map with color coded PoA coverage (4G, 5G, & Wi-Fi).

Configuration Panel: Pick from pre-built simulated networks scenario, each shows on a Map.

UE Panel: Add/remove/pause simulated devices. Types: pedestrian, vehicle & high-velocity.

MEC app instance manager Panel: Create a MEC Application Instance ID for your app and register it with the MEC platform.

Try it area: Lists of all available MEC service APIs for your selected scenario. Each entry shows the base URL you should use to call that API from browser (Swagger) or your own app.

API console: displays a table of real-time requests and notifications that the MEC Service APIs receive and generate.

ETSI The Standards People

ESTIMED enabling standard IoT & MEC deployments

Part 2: oneM2M CSE

Architecture, Two Opensource CSE Implementation (TinyIoT & ACME)

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" that receives data, routes messages, 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 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.

What is the ACME CSE?

ACME oneM2M CSE is an implementation of a *Common Service Entity* that supports a subset of the **oneM2M IoT standard**.

Designed for **educational purposes and small trials,** easy to install and run with just a few commands.

Written in Python: install anywhere Python runs

*Lead By **Andreas Kraft (JK Consulting)***

Useful links:

- *Full documentation*
- *Setup and running*
- *Github*
- *ACME CSE short video series*

Rich Text UI

Inspect resources, requests & status in the terminal

Basic Console UI

Ideal for log and debug output

Web UI

For environments without terminal access

Installing the ACME CSE

Pre-Requisite: Python 3.11 or newer

Method 1 — pip (recommended)

```
$ python -m pip install acmecse
```

To upgrade:

```
$ python -m pip install --upgrade acmecse
```

Method 2 — Manual (GitHub)

```
$ git clone https://github.com/ankraft/
    ACME-oneM2M-CSE.git
$ cd ACME-oneM2M-CSE
$ pip install -r requirements.txt
```

Use a virtual env (e.g. PyEnv) with exact versions from requirements.txt.

Starting the CSE

If installed using pip install

```
$ acmecse
```

If installed Manually (in the installation folder)

```
$ python -m acme
```

First launch? An interactive onboarding wizard guides you through initial setup and creates your config file automatically.

Configuring ACME CSE: Only Change What You Need

Works out of the box with sensible defaults. Configuration is optional, only tweak when you need something specific.

① acme.ini.default (don't edit this)
Factory defaults. Every possible setting listed with its default value, never modify directly.

② Your acme.ini (your personal overrides)
Create in the same folder. Copy only the lines you want to change. Your settings override the defaults, everything else stays at its default value.

③ The wizard does this for you on first run
An interactive setup wizard runs on first launch, asks questions and writes your acme.ini automatically. No manual editing needed to get started.

Common settings you might want to change:

Setting	What it controls	Example value
[cse] / cseName	Human-readable name shown in UIs	MyCSE
[server.http] / port	HTTP port the CSE listens on	8080
[database] / type	File-based (default) or in-memory for testing	tinydb

Full config docs: https://acmecse.net/setup/Configuration-introduction/

What is the TinyIoT CSE?

tinyIoT oneM2M CSE is a secure, fast, lightweight oneM2M server written in C, Minimal RAM and Maximum compatibility.

Designed to run on **real embedded hardware**

Written in C **Fast execution, minimal memory runs**

SQLite

Zero-config built-in database

Vue Dashboard

Web UI included out of the box

Lead By : Jieun Lee (je.lee@sejong.ac.kr), JaeSeung Song (jssong@sejong.ac.kr)

Software Engineering and Security Lab (SESLab)

Sejong University

Useful links:

- *Full documentation in Github*

Installing the tinyIoT CSE(I)

Pre-Requisite: Ubuntu 22.04 (or WSL on Windows) · Git · GCC build tools · OpenSSL

1 Install Git & build tools

You need Git to clone the repo and GCC to compile the C source code.

```
$ sudo apt install git build-essential openssl
  libssl-dev
```

2 Clone & navigate into the source

Download the source from GitHub

```
$ git clone https://github.com/seslabSJU/tinyIoT
$ cd tinyIoT/source/server
```

3 Build the server

Compile with make. A green 'server' binary appears when complete.

```
$ make
```

4 Run tinyIoT

Default: http://127.0.0.1:3000 · **Edit config.h before building to set port or enable MQTT.**

```
$ ./server
```

⚙ config.h : set port, CSE name, SERVER_TYPE (IN_CSE / MN_CSE), or enable MQTT / WebSocket before building.

📈 Vue Dashboard: open http://127.0.0.1:3000 after starting. Browse resources, inspect data, no terminal needed.

Configuring config.h

```
#define SERVER_TYPE  IN_CSE
#define SERVER_IP    "127.0.0.1"
#define SERVER_PORT  "3000"
#define CSE_BASE_NAME "TinyIoT"
#define DB_TYPE  DB_SQLITE
// #define ENABLE_MQTT
// #define ENABLE_WS  1
```

Key	Values / notes
SERVER_TYPE	IN_CSE (cloud) or MN_CSE (edge gateway)
SERVER_IP / PORT	Bind address and port (default 127.0.0.1:3000)
CSE_BASE_NAME	Human-readable name of this CSE
DB_TYPE	DB_SQLITE (default) or DB_POSTGRESQL
ENABLE_MQTT	Uncomment to activate MQTT support
ENABLE_WS	Uncomment to activate WebSocket support

Database

SQLite (default)

Zero setup — data in data.db.
Use DB_TYPE DB_SQLITE. Perfect for dev and small deployments.

PostgreSQL (production)

`sudo apt install postgresql libpq-dev && sudo systemctl start postgresql` then set: `#define DB_TYPE DB_POSTGRESQL`

```
CREATE DATABASE tinydb;  CREATE USER tinyuser WITH PASSWORD 'tinypass';  GRANT ALL PRIVILEGES ON DATABASE tinydb TO tinyuser;
```

Protocol Extensions

MQTT Install: `sudo apt install mosquitto && sudo systemctl start mosquitto`
Enable: `#define ENABLE_MQTT` QoS 0/1/2 · keepalive 60s · client ID: TinyIoT

WebSocket Install: `sudo apt install libwebsockets-dev`
Enable: `#define ENABLE_WS 1` host: localhost · port: 8081