ETSI White Paper No. 20

Developing Software for Multi-Access Edge Computing

2nd edition – February 2019

ISBN No. 979-10-92620-29-0

Authors:

Dario Sabella, Vadim Sukhomlinov, Linh Trang, Sami Kekki, Pietro Paglierani, Ralf Rossbach, Xinhui Li, Yonggang Fang, Dan Druta, Fabio Giust, Luca Cominardi, Walter Featherstone, Bob Pike, Shlomi Hadad

ETSI
06921 Sophia Antipolis CEDEX, France
Tel +33 4 92 94 42 00
info@etsi.org
www.etsi.org

About the authors

Dario Sabella
Intel

Vadim Sukhomlinov
Intel[1]

Linh Trang
Sony

Sami Kekki
Huawei

Pietro Paglierani
Italtel

Ralf Rossbach
Intel

Xinhui Li
VMware

Yonggang Fang
ZTE

Dan Druta
AT&T

Fabio Giust
Athonet

Luca Cominardi
UC3M

Walter Featherstone
VIAVI Solutions

Bob Pike
ACS

Shlomi Hadad
Saguna

[1] The work was done when the author was with Intel. Now he is working with Google.

Contents

Introduction

Edge Computing refers to a broad set of techniques designed to move computing and storage out of the remote cloud (public or private) and closer to the source of data. For the emerging class of "5G Applications" this is often a matter of necessity. Locating such applications in a traditional cloud does not allow one to meet certain stringent requirements, such as roundtrip latency. In other cases, such as the Internet of Things (IoT) and Vehicle to everything communication (V2X), the amount of data is expected to increase rapidly. Edge computing can mitigate this by collecting and processing data closer to the user.

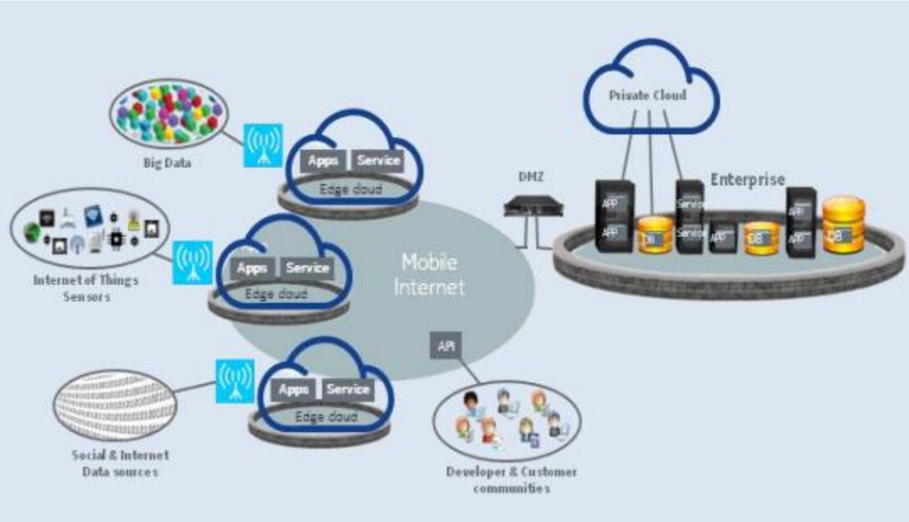

Big Data
Apps
Service
Edge cloud
Internet of Things Sensors
Social & Internet Data sources
Mobile Internet
API
Developer & Customer communities
DMZ
Private Cloud
Enterprise
APP
Service
DB

Figure 1: Overview of the MEC system (see references [1][2])

ETSI Industry Specification Group (ISG) MEC (Multi-access Edge Computing) focuses on enabling edge computing at the access network (mobile or otherwise), thus bringing edge computing as close as possible to the user without it being in the user device. The group was established in September 2014 to standardize APIs that will enable application and content providers to utilize computing capabilities present at the edge of the network. MEC enables successful deployment of new use cases and various services that can be customized according to the customer requirements and demands. Some of key applications and use cases are:

- Video content delivery optimization
- Video stream analytics and video surveillance
- Augmented Reality and Virtual Reality (AR/VR)
- Enterprise applications enablement and local breakout
- Applications with critical communication needs such as traffic safety and control, autonomous cars, Industrial IOT and Healthcare
- Connected Cars
- IoT applications and Gateway
- Location and Context aware Services
- Smart City applications

The current prevalent distributed computing software development model uses a client-side to initiate server requests and a remote server-side to process these requests (the client-server model). This allows application developers to take advantage of centralized compute and storage and has been a major driver of the emergence of cloud computing. However, for MEC applications, developers need to identify features of their applications that require processing at the edge as distinct from features that require high compute power or that do not require near real-time response and can, therefore, be deployed at a central location. Applications have to be designed in a way which supports distributed processing, synchronization of contexts, multi-level load-balancing.

This idea is quite recent, although not totally new, and the ecosystem is quickly moving to use systems like Greengrass for Amazon's AWS Lambda, Microsoft's Azure IoT stack and GE's Predix to enable it. Let's take, for example, AWS Greengrass. This consists of the AWS Greengrass core (which is responsible for providing compute capabilities closer to the devices) and the AWS IoT devices enabled with AWS IoT Software Development Kit (SDK). Using this architecture, AWS IoT applications can in real time respond to local events and use cloud capabilities for certain functions that don't require real time processing of data. An IoT application developer targeting AWS Greengrass has to architect the application in a way that uses these edge systems for certain features that require real time processing, or which perform some other useful tasks (e.g. limiting the data flood to the central location), while keeping other features in the traditional cloud.

To provide these new services and to make the most out of MEC it is also important for the application developers and content providers to understand the main characteristics of the MEC environment and the additional services which distinguish MEC from other "edge computes", namely: extreme user proximity, ultra-low latency, high bandwidth, real time access to radio network and context information and location awareness.

On this basis this white paper provides guidance for software developers on how to properly approach architecting and developing applications with components that will run in edge clouds, such as those compliant with ETSI's MEC standards. The white paper will summarize the key properties of edge clouds, as distinct from a traditional cloud point-of-presence, as well as the reasons why an application developer should choose to design specifically for these. It will then provide high-level guidance on how to approach such design, including interaction with modern software development paradigms, such as micro-services - based architectures and DevOps.

The need for an evolved approach

MEC offers to application developers and content providers cloud-computing capabilities and an IT service environment at the edge of the network. Consequently, MEC introduces a standard for supporting an emerging cloud paradigm for software development communities. In fact, up to now a "traditional" client-server model of application development has been the dominating approach to developing applications for at least 2 decades. The emergence of edge computing, e.g. MEC, evolves this environment, by introducing an intermediate element at the network edge.

A MEC point-of-presence (PoP) is distinct from a traditional cloud PoP. It may offer significant advantages to application components/services running there, while also presenting some challenges, e.g. higher cost, relatively small compute footprint, good local but not global reachability, etc. As such, it is crucial for an application developer to design with specific intent towards running some application components at the network edge when developing for MEC.

This results in a new development model with 3 "locations": Client, Near Server, Far Server (depicted in Figure 2). The client location can be a traditional smartphone or other wireless connected compute elements in a car, smart home or industrial location that can run dedicated client applications. The model is quite new to most software developers, and while modern development paradigms (e.g. microservices) make it easier to adapt to it, a clear and concise summary of this new development model and guidance on how to properly approach it will help accelerate the application development for the network edge and thus accelerate MEC adoption.

As depicted in Figure 2, a MEC Host, usually deployed at the network edge, contains a MEC platform and the compute, storage and network resources for applications in VMs or containers. The MEC platform offers a secure environment where MEC applications may, via RESTful APIs, discover, advertise, consume and offer services.

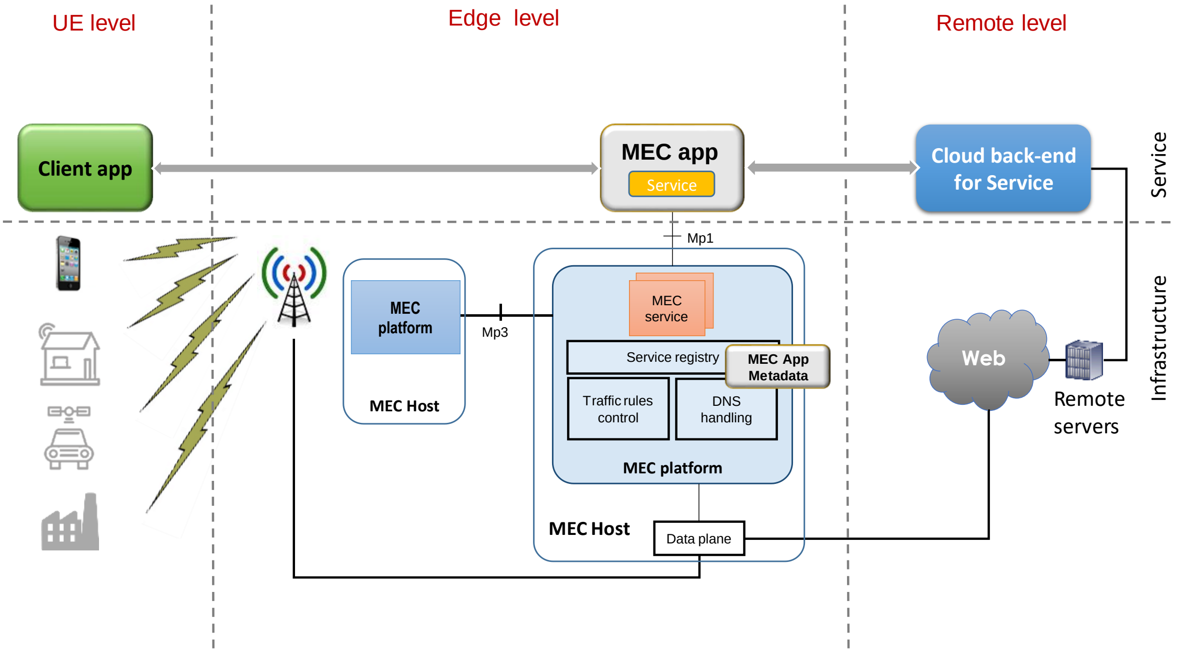


Figure 2: New application development paradigm introduced by MEC.

Designing with the Edge in Mind

A key principle of modern system design, known as the hour-glass model (and realized with IP as the single protocol at the waist of the hour-glass) is that applications and networks should be completely agnostic to each other. This has been key to the success of the Internet and the millions of applications running over it. It allows applications to run over any IP-based network, while any network can use IP to support pretty much any IP-based application. Nevertheless, when it comes to actual application performance, both aspects (the application design and the network) are important and remaining completely agnostic becomes difficult. To date, few traditional Internet applications encountered issues with this approach since the requirements of applications have traditionally been much looser than the performance that networks can deliver. This is about to change. Already practices are emerging that consider network aspects as part of application design. A widely used example is adapting the application behaviour to throughput limitations in the network, e.g. adjusting the video stream compression ratio in response to throughput throttling, while also considering the application state, e.g. whether the application is active or idle or in suspended state, etc. Nonetheless, network conditions and topology characteristics are typically considered during the software design phase as an environmental input out of the programmer’s control and the application *passively* adapts to these.

The environment enabled by the MEC platform is where the network and the applications can converge in a meaningful way without giving up the key benefits of the hour-glass model. MEC can support any application and any application can run in MEC. However, MEC can offer additional services to those applications which have been designed to be MEC-aware. MEC Application Enablement (described in ETSI GS MEC 011 [3]) introduces such a service environment, and this can be used to improve the user experience tremendously. Software designed to take advantage of MEC services can leverage additional information about where the application is supposed to run, in terms of expected latency, throughput and other available MEC services. Simply put, with MEC, the environment becomes *less unpredictable* and environmental (i.e. contextual) information can be leveraged to *actively* adjust the application behaviour in run time. This means that the network characteristics can factor in during the design of the end to end service. For example, through the MEC Radio Network Information (RNI) API, it is possible to precisely monitor the radio link, and this information can be obtained by a MEC application that uses it e.g., to drive the behaviour of the client application in the user device, as well as of the application in a central cloud. Similarly, a MEC application can make a bandwidth request using the Bandwidth Management API to reserve networking resources in the MEC system (more details about MEC APIs can be found later in this paper. This allows edge applications to benefit from low latency and high throughput in a predictable/controllable way, and this information can be leveraged at the time of service design to optimize the end-to-end service architecture. In addition, the network itself could also benefit from the MEC services provided by the applications, for instance the network scheduler could also predict the incoming user behaviour to maximize the network efficiency.

From the discussion so far, it emerged already a key aspect of software design for MEC: the end-to-end service can be split into three applications or components: *terminal device component(s)*, *edge component(s)* and *remote component(s)*. This concept should not be confused with the traditional software modularization, but rather seen as a distribution of components to leverage different features of the computing environment. In fact, if the former considers the division of tasks to improve the development and maintenance of the code, the latter is based on distributed computing to meet specific performance figures achievable only at the network edge. In many current services, the different components either run in the same data centre or are sparsely distributed at unpredictable locations (e.g.

P2P applications), and software instances are scattered mostly to address load balancing. Software modularization is not only possible in MEC, but also encouraged to assign execution tasks at the most appropriate location. In particular, as discussed below, a microservices-based architectural approach is particularly well suited for MEC.

This aspect creates an additional paradigm with respect to a more traditional client-server architecture, since an additional processing stage (at the edge) must be added to the application's workflow, with well-defined characteristics and capabilities. For instance, the terminal device can do some preliminary processing to determine the need for further actions. Such preliminary processing requires near zero latency and it requires the terminal device to support some computing capabilities, e.g. to receive and instantiate algorithms or instructions. The edge component(s) include a set of operations that the application performs at the edge cloud, e.g. to offload the computing away from the terminal device while still leveraging very low latency and predictable performance, or offloading high bandwidth load from the network backbone, or extracting some information using RNI API or Location API. The remote components implement operations to be carried out in the remote data centre, e.g. to benefit from large storage and database access.

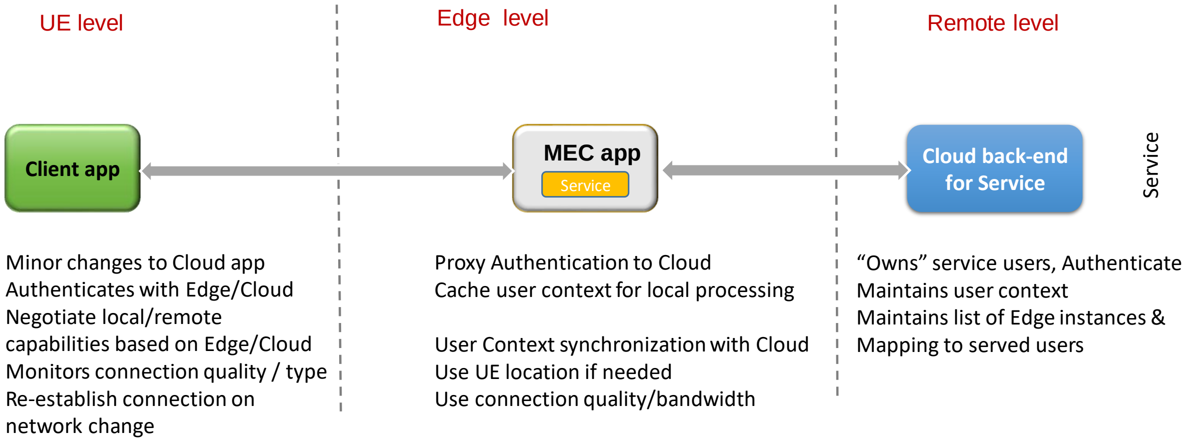

UE level
Edge level
Remote level
Client app
MEC app
Service
Cloud back-end for Service
Service
Minor changes to Cloud app
Authenticates with Edge/Cloud
Negotiate local/remote capabilities based on Edge/Cloud
Monitors connection quality / type
Re-establish connection on network change
Proxy Authentication to Cloud
Cache user context for local processing
User Context synchronization with Cloud
Use UE location if needed
Use connection quality/bandwidth
"Owns" service users, Authenticate
Maintains user context
Maintains list of Edge instances &
Mapping to served users

Figure 3: Example of splitting an application into "terminal", "edge" and "remote" components

What is worth highlighting here is that MEC can be exploited to implement computation offloading techniques among all the application's components. In fact, the server can be programmed to dynamically shift the processing among the terminal, the edge and the remote component(s), for a number of reasons ranging from adaptation to network conditions, improving application specific KPIs, policies, costs, etc. The processing distribution may be driven by certain performance objectives, e.g. providing the best user experience.

Another aspect inherent to MEC is that service providers can exploit the geographical distribution to serve different user populations and thus tailor their service knowing the peculiarities of the covered area. For instance, media content can be adapted to the linguistic and cultural characteristics of a given area, or customized advertising can be tailored to the needs of local businesses. Content Delivery Networks (CDNs) are only one, but perhaps the most straightforward example of a use case that benefits from geospatial distribution. Nevertheless, most of content provided by today's CDNs is not as delay sensitive as the services from other MEC applications, e.g. for virtual/augmented reality, for which it is crucial to run at a specific location to meet the application's stringent latency requirements.

Architecting and Developing for the Edge

This section describes how to deploy an application in a MEC system. Some features are fully defined by the MEC standard, whereas others are left to the specific implementation of the MEC provider.

As a first step there are two APIs that the developer should consider, specifically Mx2 API (see ETSI GS MEC 016 [4]) and Mp1 API (see ETSI GS MEC 011 [3]). Mx2 allows a device application to interact with the MEC system and is further described later in this paper. Mp1 is the reference point between MEC applications and the MEC platform, which allows these applications to interact with the MEC system by discovering, advertising, consuming and offering MEC services. In addition, the application may influence the traffic routing by updating the MEC traffic rules. There are also specific service-related APIs such as Radio Network Information API (ETSI GS MEC 012 [9]) and Location API (ETSI GS MEC 013 [10]). Depending on whether the application wishes to consume MEC services, or even produce them, the developer may also want to consider these service-related APIs.

To facilitate MEC application design, MEC communications can be divided in phases (described in detail in the following sub-sections):

- Phase 1 – MEC application packaging & on-boarding
- Phase 2 – MEC application instantiation
- Phase 3 – communication between client-side app and MEC app
- Phase 4 – usage of the MEC platform and services

These phases are described in detail in the following sub-sections. Before analysing them in detail some preliminary high-level considerations for when developing and deploying an application in MEC are provided:

- **DNS-based solution**

  The application must be designed to support a DNS-based solution for traffic redirection. The application sends a DNS query for a registered domain name.

- **Domain name**

  Register a name for the service (FQDN), which is known to client application to gain access.

- **Cloud back-end**

  If there is a requirement for the service to be available regardless of whether a local MEC system exists or not, then a back-end service will be required as a fall-back solution hosted in the cloud or at alternative premises.

- **Sensitive user context data**

  User context data may be sensitive from a legal point of view and may not be allowed to be transferred from one jurisdiction to another. This use context transfer is something the developer must handle in their application if application mobility is to be supported.

- **Packaging the application**

A MEC application runs as a virtualized application, such as a virtual machine (VM) or a containerized application, on top of the virtualization infrastructure provided by the MEC host. Appropriate packaging of the application is inextricably linked to the run-time environment of the MEC system and therefore the developer needs to adapt accordingly, providing the package e.g. as VMs, Docker containers or Kubernetes templates. This is further described in the next section

- **Provide meta-data with application requirements**

The application needs to be provided with its requirements such as latency tolerance, network resources, storage resources, CPU etc. that the MEC system needs to account for. These requirements are provided in the Application descriptor (see ETSI GS MEC 010-2 [5]), which forms part of the application package. Information on whether the application produces a service or is dependent on other services must also be included.

Phase 1: MEC application packaging & on-boarding

MEC applications are packaged by application developers (or in some cases also by MEC operators), and typically set up as a VM or Container (e.g. qcow2/ vmdk images, Docker containers, etc.) with all the necessary dependencies according to a specific MEC platform's requirements and configurations. For security reasons application providers usually sign their application package before sending it to the OSS for set up purposes. Various options exist to package and onboard applications within a MEC environment which likely will have contractual considerations between the interested parties. For this reason, the specific details of application **onboarding** are not defined by the standard and are rather left within the platform and service provider domain. However, the high-level details and steps are provided below.

Figure 4 highlights the entities involved in application onboarding. When the OSS receives requests for managing of applications (e.g. onboarding, instantiation or termination) it makes the decision on whether to grant these requests, or not. Granted requests are forwarded to the MEC Orchestrator (MEO) for further processing. After receiving a request from the OSS, the MEO has the responsibility of onboarding MEC applications into MEC systems, including checking the integrity and authenticity of the signed packages, validating application rules and requirements and if necessary, adjusting them to comply with operator policies, keeping a record of on-boarded packages, and preparing the virtualization infrastructure manager(s) to handle the applications.

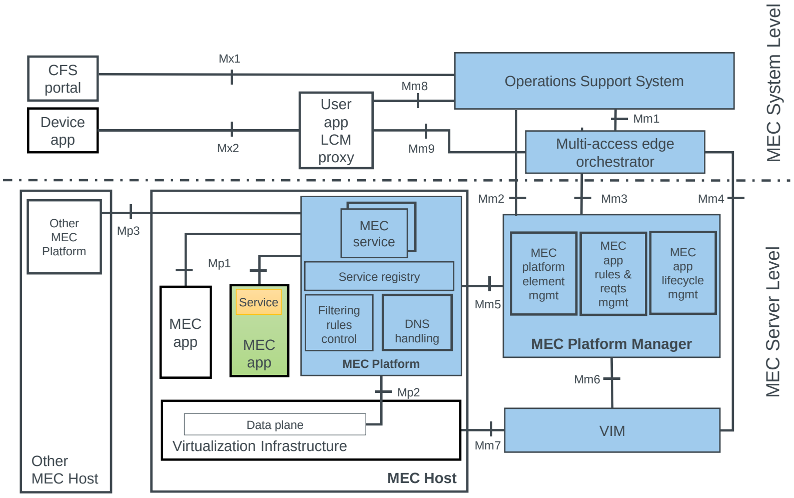


Figure 4: Entities involved in Application on-boarding, instantiation and operation

The MEO assigns an application package ID and provides the MEC Platform Manager (MEPM) with the location of the application image if it is not yet on-boarded in the MEC system. The MEPM prepares the Virtualized Infrastructure Managers (VIMs), selected by the MEO for application instantiation, by providing the necessary infrastructure configuration information and sending the application images, which are then stored by the VIM.

Once on-boarded, the application package is in "Enabled, Not in use" state. From there further application lifecycle management actions may be performed by the OSS in response to application package enable, disable, query and deletion commands.

Phase 2: MEC app instance instantiation and operation

In this section we describe how the developer can get the MEC application instantiated in the MEC system and then make it available to the target audience (the final customers, or in general the app users).

Application initialization may be triggered from a device or from the Operation Support System (OSS). With the first option (figure 4 below), a developer is able to interact directly with the MEC system using a device with a client supporting the Mx2 API (see ETSI GS MEC 016 [4]). In MEC parlance such a client is referred to as a device application. This also requires the MEC system to support the MEC standards defined *UserApps* feature. The User Application LifeCycle Management Proxy (UALCMP) exposes the Mx2 API to the device application (see Figure 4). It allows the device application to request the following application lifecycle management operations from the MEC system: query the available applications, instantiation and deletion of an application and update of an existing application context.

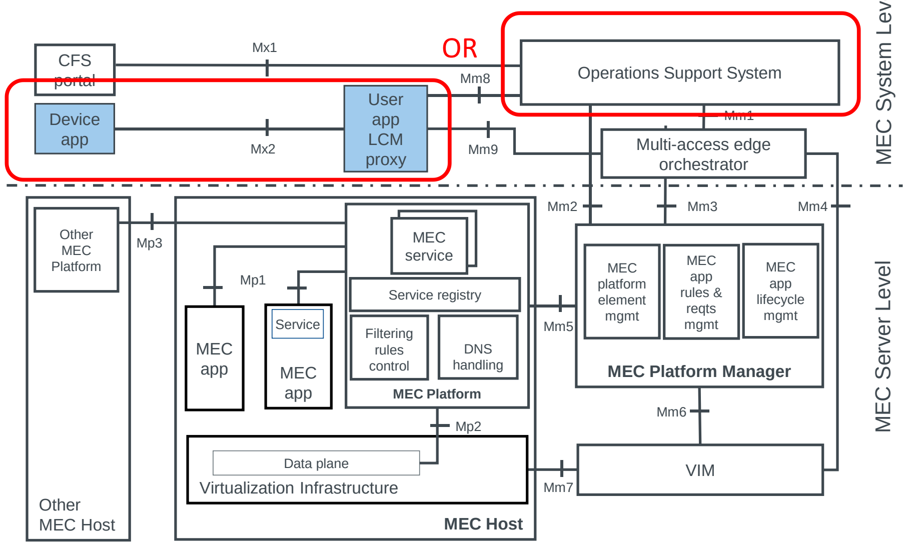


Figure 5: MEC app instantiation options

The other option for the application instantiation is via direct interaction between the developer/service provider and the MEC operator without the involvement of a device application. The MEC operator then triggers the application on-boarding and instantiation directly through the OSS. With this approach the application developer does not have to concern themselves with this initial phase.

The following two subsections describe the instantiation process from a client (external) perspective and give a brief overview of how this is performed from a MEC platform (internal) perspective.

The client perspective

In real life scenarios, for direct interaction with the MEC system, the device application needs a subscription or some other contract with the MEC operator[2]. The subscription authorizes the device application to make use of the MEC services. The ETSI MEC specification relies on OAuth 2.0 in securing the device application's access to MEC system resources. The device application first authenticates and then gets authorized in operator's AA server, which will provide the necessary credential for Mx2 API operations, i.e. the access token. Once it has those and has discovered the UALCMP it can then invoke Mx2 API. In each API call the device application is expected to include the access token in the "Authorization" header field as a bearer token as defined in IETF RFC 6750 [6]. The exact methodology for the device application to acquire the token is beyond the scope of the MEC specification. It is also noteworthy that any given MEC operator may support other means of authentication and authorization, which are also beyond the scope of MEC standard.

With the Mx2 API there are different application instantiation options. If the application image is already available in the MEC system, the developer can request it to be instantiated directly. The Mx2 API also supports querying of all available applications and then pick the one to be instantiated. If, however, the application image is not yet in the MEC system, the developer can provide the MEC system with a link to the application package that contains the image and the descriptor. Using this link, the system can retrieve and then instantiate it. In this latter option, if at all supported by the MEC operator, the image repository most likely will have to be trusted by the operator, or the operator may even have a repository of its own where the developers can make their application images available. In both options the MEC system's response to a successful instantiation includes the address of the instantiated application. The attribute *ReferenceURI* in the Application Context in the response body from the UALCMP conveys the address of the application instance. Once the device application has received the address for the MEC application, the developer can disseminate it among the target audience through their own chosen means. During the lifetime of the application instance the device application will receive notifications of the change, if any, of the application address.

According to the ETSI MEC terminology and specifications, a MEC application that was instantiated in the MEC system in response to a request of a user over Mx2 API is called a user application. In addition to instantiation, the Mx2 API also supports the deletion of the user application. The developer can request the application context to be deleted, resulting in a termination of the application and removal of the application's resources in the MEC system for application context in question.

At this point it is worth emphasizing a couple of essential aspects on the application instantiation in MEC:

1. All operations between the authorized device application and the system proxy (UALCMP) on Mx2 API are management plane operations. There the application instance is *referred to* via the identifier of the context that was created by the MEC system for this application instance. Data type AppContext in ETSI GS MEC 016 [4] represents the information on application context, and there the attribute *contextId* is the identifier of the said application context. Each application instance present in the MEC system has their own unique context ID on Mx2.

2. On the user plane any user with a connected device can attempt to contact the application by using the application's address (the one originally found in the ReferenceURI, e.g. an IP address) as is the case with any internet application. As the MEC system can also configure the DNS server, the application instance may also be addressed by its FQDN (Fully Qualified Domain Name), if

[2] The MEC operator may be different from the Mobile Network Operator (MNO).

available. Any security, authentication, authorization, etc. related to accessing the instantiated application on the user plane is beyond the scope of the MEC specification and will have to be provided by the application/service provider.

3. The developers requesting an application to be instantiated cannot select the location where the application is instantiated, but they shall describe the requirements of the application. These requirement attributes are defined in the Application descriptor data type in ETSI MEC GS 010-2 [5] and they include the C/S/N requirements but also the maximum latency tolerated by the application. It is then the MEC system Management & Orchestration (MANO) that selects the ideal location for the application instance. For an application image made available in a repository for the MEC system to fetch it, the application descriptor is expected to be included in the application package.

The MEC platform perspective

As explained earlier, the instantiation of a MEC application on a MEC host is initiated by the OSS, e.g. in response to a request from the device application, or via direct interaction between the service provider and MEC operator. The OSS forwards the granted instantiation to the MEO, which is responsible for oversight of available MEC apps/services, and app termination. The MEO selects the target MEC platform (MEP) and forwards the request to selected MEPM, which configures the MEP accordingly.

An instantiation request contains information about the application to run, application rules and requirements and possibly other information, such as the target location where the application is to be deployed. MEPM can send lifecycle requests to the VIMs for allocating virtualized resources (compute, storage and networking) and for instantiating the MEC application. Once set up, the MEC application is enabled to interact with MEP over Mp1 to perform certain support procedures related to the lifecycle of the application, such as indicating availability, preparing relocation of user state, etc. MEPM also receives virtualized resources fault reports and performance measurements from VIM for further processing and updates the initial set-up information accordingly.

After an application has been instantiated, the following operations can be performed on the application instance through the application lifecycle management APIs (see ETSI GS MEC 010-2 [5]):

- Start: instruct the application instance to run and produce the service
- Stop: instruct the application instance to stop running and producing service

When an operation on the application instance is invoked, the application lifecycle management will trigger a special task, i.e. Application lifecycle Operation Occurrence, to track the operation and send a notification to the application lifecycle management once the operation finishes. The application lifecycle management then updates the operational state of the application instance accordingly.

The MEC platform and MEC applications or services can subscribe to the notification on application instance operational state change through the application lifecycle management APIs.

Phase 3: client-side app and MEC app communication

A device application is logically separate from the actual client application (see Figure 6) which is the one requesting services from the MEC application. Any end user device may have the client application installed. A client application most often is unaware of the edge deployment of the server application, i.e. the MEC application. The device application, on the other hand, is needed for invoking user application lifecycle management operations on MEC system’s management plane, as explained in phase 3.

There are two ways for the client application to connect with a MEC application instance: The first option is that the developer/service provider takes responsibility for making the address of the MEC application, which it receives from the MEC system, available to potential client applications. The other option is that the client application discovers the MEC application instance via DNS look-up. If the domain name is owned by the application provider/developer, the developer needs to ensure that the domain name is managed and updated in the corresponding DNS server. The MEC platform supports the handling of MEC applications' DNS records accordingly.

Depending on what functionalities are supported in the MEC system(s) where the application is being deployed the developer needs to support at least one of the two options above.

Figure 6 depicts the client-side application components; client application(s) and the device application, as well as the relevant MEC system components and the ways how the client-side application components can communicate with their MEC side peers. MEC platform has been authorized to configure MEC application's DNS records for a successful address resolution.

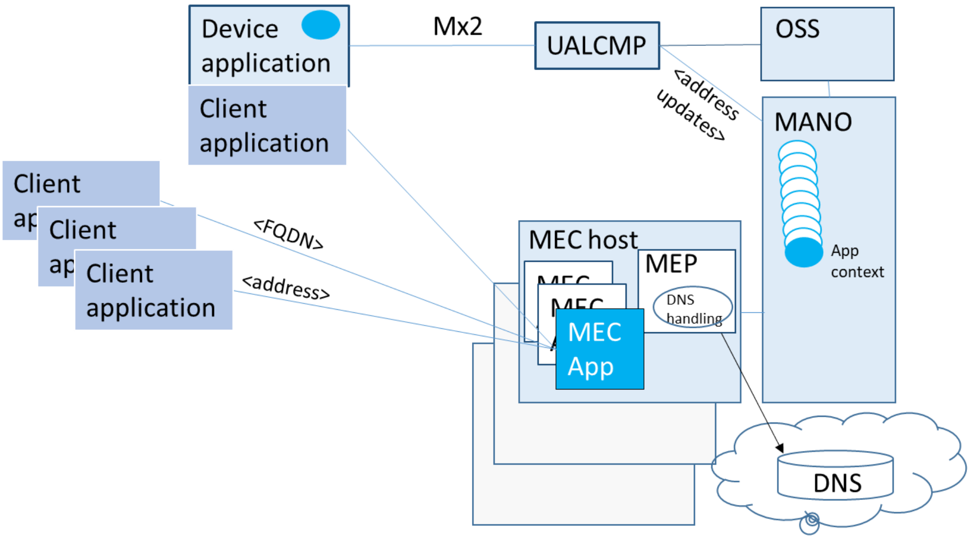


Figure 6: Communications between terminals and the MEC system

Phase 4: usage of the MEC platform and services

Once the MEC application is up and running and becomes operational, it can consume MEC services. These services may be produced by the MEC platform or be produced by another MEC application. MEC services (e.g. RNI, Location, Bandwidth and UE Identity) are available for the MEC platform and authorized MEC applications.

RESTful design

MEC specific APIs are built upon RESTful APIs. The concept of RESTful programming is widely accepted in the industry and many developers are already familiar with the principles of these APIs. A RESTful API is an application program interface (API) that uses the HTTP protocol as a tunnel or transfer mechanism for interaction between remote entities. RESTful refers to a stateless design through REpresentational State Transfer (REST), an architectural style popular for development of web services ([7]).

ETSI's design principles for developing RESTful MEC APIs are outlined in ETSI GS MEC 009 [11], along with http methods, templates, conventions and patterns to create RESTful APIs for MEC. MEC APIs are not only documented in ETSI specifications, they are also available in YAML and JSON format in a Git repository at https://forge.etsi.org/, presented via OpenAPI compliant descriptions.

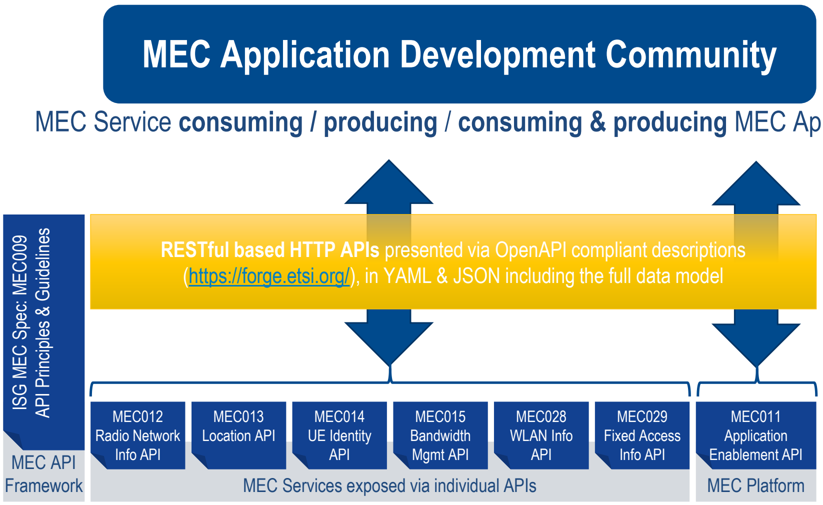

MEC Application Development Community
MEC Service consuming / producing / consuming & producing MEC Apps
RESTful based HTTP APIs presented via OpenAPI compliant descriptions
(https://forge.etsi.org/), in YAML & JSON including the full data model
ISG MEC Spec: MEC009
API Principles & Guidelines
MEC API Framework
MEC012 Radio Network Info API
MEC013 Location API
MEC014 UE Identity API
MEC015 Bandwidth Mgmt API
MEC028 WLAN Info API
MEC029 Fixed Access Info API
MEC011 Application Enablement API
MEC Services exposed via individual APIs
MEC Platform

Figure 7: Phase 5 - overview of the MEC Platform and Services from a developer perspective

Radio Network Information API

MEC applications run at the edge of the network where the environment is characterized by low latency, proximity, high bandwidth and exposure to location and up-to-date radio network information. While low-level information on current radio conditions is available in the Radio Access Network (RAN), the MEC server is in relative proximity and connected to the base station. The Radio Network Information Service (RNIS) is a service that provides radio network related information to authorized MEC applications and the MEC platform. The provided information originates from the RAN, based on contents defined in 3GPP specifications. The granularity of Radio Network Information desired by the application can be configured per cell, per UE, per QCI/5QI class, or may be requested over a period of time only. Some of the typical information that can be exposed is listed below:

- Up-to-date information about radio network conditions.
- Layer 2 measurement and statistics information based on KPIs of the user plane, according to 3GPP specifications.
- Information about UEs connected to the radio nodes associated with the MEC host, their UE context, related radio bearers, Quality of Service (QoS) information, throughput, neighbour cells, etc.
- Changes on the information above, based on API subscription, for example to provide notifications on cell change, radio bearer release or reconfigurations, updates to carrier aggregation configuration, UE measurement reports, related to UEs connected to the radio nodes associated with the MEC host.

The Radio Network Information (RNI) may be used by MEC applications and/or the MEC platform to optimize the existing services and to provide new services that are based on up-to-date information on radio conditions. For example, RNI can be used to include bitrate recommendations for video or voice calls based on actual real-time throughput available for a specific connection, or the MEC platform can utilize RNI to optimize mobility procedures required to support service continuity. RNI is also used to optimize network operation. The RNIS supports a wide range of use-cases, some of which are described in ETSI GS MEC 002 [12].

The service consumers interact with the RNIS over the RNI API to obtain contextual information from the radio access network. The Radio Network Information API supports both queries and subscriptions (pub/sub mechanism) that are used over the RESTful API or over the message broker of the MEC platform. More information about the RNIS as well as the APIs is available in ETSI GS MEC 012 [9].

Location API

The availability of accurate location information is crucial to a number of services and enables area-specific application data content. MEC's Location Service (LS) can provide location-related information to a MEC platform or authorized applications.

The LS leverages the Zonal Presence Service described by Small Cell Forum in [16] and [17]. The Location Service is accessible through RESTful APIs originally defined by Open Mobile Alliance (OMA). The incorporated API definitions as well as the full description of the location service are available in ETSI GS MEC 013 [10]. The Location Service is registered and discovered over the Mp1 reference point defined in ETSI GS MEC 003 [13].

Consumers of the Location Service may use the LS API to obtain the location information of a UE, a group of UEs, or the radio nodes associated with a MEC host. For example, the service can perform active device location tracking or provide location-based service recommendations to allow a variety of additional services tightly coupled with a specific place (such as a shopping mall). It is possible to report information such as the distance between a specified UEs or the distance between a specified location and a UE, provide a list of UEs in a particular area of location, or even report when specific UEs move in or out of a particular area.

The service supports both geolocation, such as geographical coordinates, and logical location, such as a Cell ID. Subscriptions to location information are also offered, including periodic location information updates, updates on changes in distance and location updates relating to UEs in a particular area of location, and more. The Location Service API supports both queries and subscriptions (pub/sub mechanism). To facilitate collection of statistics, anonymous location reporting (without related UE ID information) can be used. Operators or third-party services can use the location data for security, safety, and data analytics, or to optimize the network.

Bandwidth Manager API

When a MEC host runs applications in parallel and is using the same network resources, applications are typically competing over available bandwidth. Data traffic associated with different MEC applications (or even specific application sessions) may have different bandwidth requirements with respect to e.g. throughput and priority. A bandwidth management service (BWMS), which likely is produced by the MEC platform, allows a fair distribution of bandwidth resources between applications.

Using the BWMS, different applications, whether managing a single instance or several sessions, may request specific bandwidth requirements for the whole application instance or different bandwidth requirements per session. The BWMS aggregates all the requests to help optimize bandwidth usage and allocates it accordingly.

The Bandwidth Management (BWM) API, described in ETSI GS MEC 015 [14], enables registered MEC applications to request a specific bandwidth allocation. Applications can call the BWM API to register, update or unregister their specific bandwidth requirements (size/priority). It is also possible for the application to query the API to get their configured bandwidth allocation. A specific MEC application or application session is identified using a set of filters within the resource request of the API. The API uses a RESTful API design. Furthermore, the BWMS design might interface with the Radio Network Information Service to use available Layer 2 and QoS information to facilitate the traffic arbitration.

UE Identity API

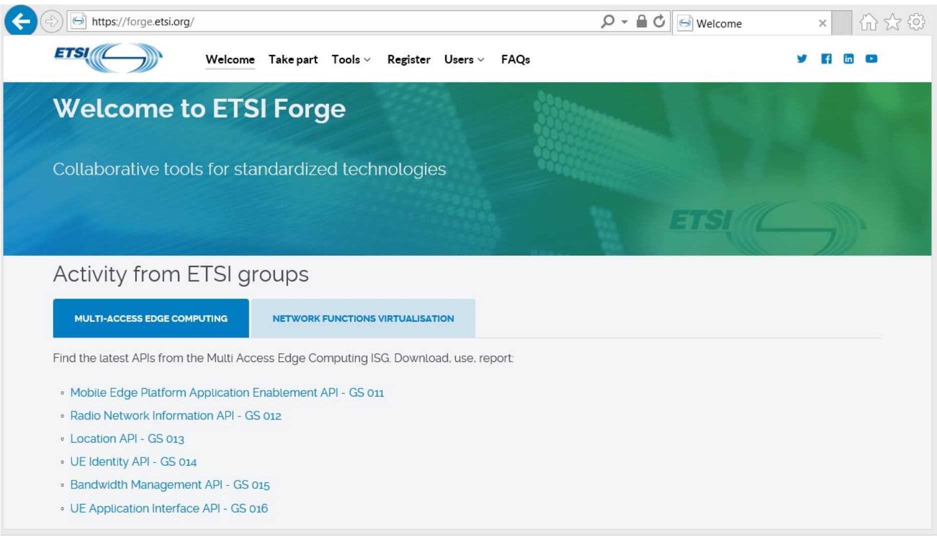
The MEC platform provides functionality to facilitate the association of IP traffic flows with a particular UE using an externally defined tag rather than the UE Identity directly. The association between a tag and the UE Identity helps preserve user privacy protect identity information at both mobile and enterprise network. A MEC application manages related access control and the integrity of the user content. This is where the UE Identity API, described in ETSI GS MEC 014 [15], comes in.

The UE Identity API provides functionality for a MEC application to register/deregister a tag (representing a UE) or a list of tags in the MEC platform. Each tag has been mapped to a specific UE in the mobile network operator’s system and the MEC platform is provided with the mapping information. The MEC platform uses registered tags to apply traffic filters rules based on that tag. This enables authorized UEs to get their user plane traffic routed directly to e.g. a local enterprise network without having to pass through the MEC application. The tag-based traffic filter rules are handled in the MEC platform and described in ETSI GS MEC 011 [3].

Building your first MEC application

The preceding sections have provided a comprehensive description of the key considerations a developer should consider when creating a MEC application. Annex C also contains a couple of exemplary use cases for MEC applications. This section provides some more practical insights on developing a "*Hello World*" style MEC application.

In order to increase the accessibility of the group's specifications, ISG MEC has published OpenAPI Specification (OAS) (https://github.com/OAI/OpenAPI-Specification) compliant descriptions of its service and Platform Application Enablement (Mp1) API specifications on the ETSI hosted Forge site (https://forge.etsi.org/), Figure 8. The OAS offers an open source framework for defining and creating RESTful APIs. YAML (or JSON) interface files compliant to the OAS are both human and machine-readable providing the means to describe, produce, consume and visualize RESTful web services. Based on the interface file clients, i.e. potential MEC applications, can understand and consume services without knowledge of server implementation or access to the server code by reading the declarative resource descriptions. A large ecosystem of tooling has been built up around the OAS, where the Swagger Editor (https://editor.swagger.io/) is of significant note. This enables server and client stubs, in a large variety of languages, to be automatically generated for each of the MEC APIs and used as the starting point in the creation of a MEC application.



Welcome to ETSI Forge

Collaborative tools for standardized technologies

Activity from ETSI groups

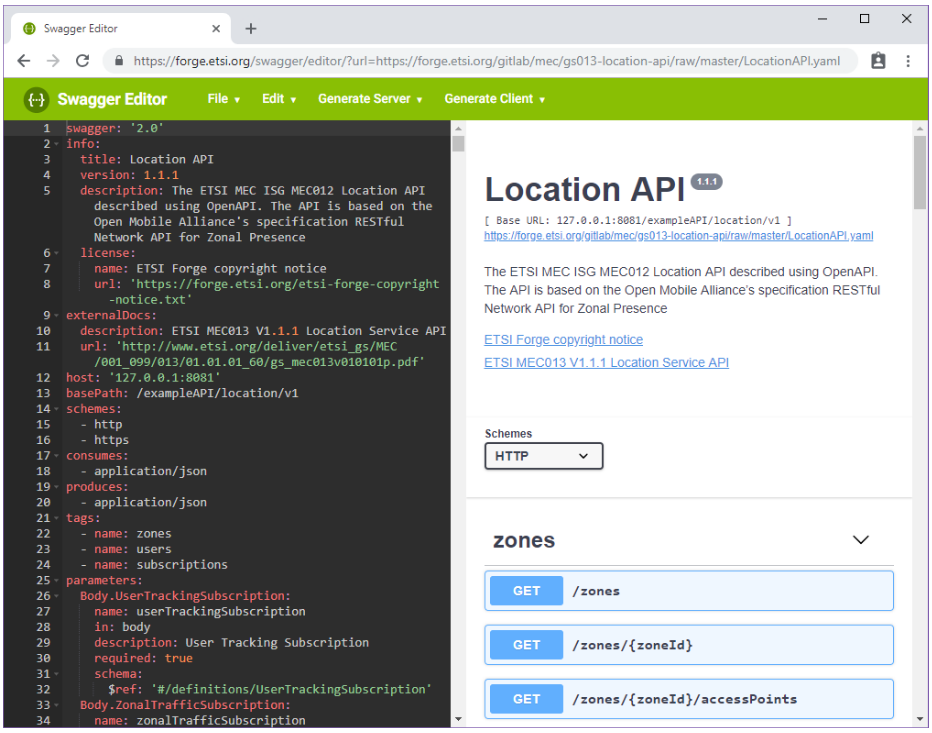
MULTI-ACCESS EDGE COMPUTING | NETWORK FUNCTIONS VIRTUALISATION

Find the latest APIs from the Multi Access Edge Computing ISG. Download, use, report:

- Mobile Edge Platform Application Enablement API - GS 011
- Radio Network Information API - GS 012
- Location API - GS 013
- UE Identity API - GS 014
- Bandwidth Management API - GS 015
- UE Application Interface API - GS 016

Figure 8: The ETSI Forge landing page

The MEC Location API can be taken as a specific example, where the latest version can be viewed within the Swagger Editor hosted on the Forge site (see Figure 9), or by pasting the YAML directly into the online Swagger Editor. As seen in the figure, this offers the option to "Generate Server" where several languages can be selected. It is likely that Cross-Origin Resource Sharing (CORS) support will need to be enabled for the server to allow resources to be requested from the client domain that will be outside the server domain from which the resource representations are served. By default, a server created using the Location API description will be listening on port 8081 of your localhost once running (reference the "host:" line in the figure). Further capability can then be added to the server. For instance, using it to interface to a database containing additional example data for the responses, rather than just using the example response content provided in the interface file.


Swagger Editor
https://forge.etsi.org/swagger/editor/?url=https://forge.etsi.org/gitlab/mec/gs013-location-api/raw/master/LocationAPI.yaml
Swagger Editor
File ▾
Edit ▾
Generate Server ▾
Generate Client ▾

```yaml
swagger: '2.0'
info:
  title: Location API
  version: 1.1.1
  description: The ETSI MEC ISG MEC012 Location API
    described using OpenAPI. The API is based on the
    Open Mobile Alliance's specification RESTful
    Network API for Zonal Presence
  license:
    name: ETSI Forge copyright notice
    url: 'https://forge.etsi.org/etsi-forge-copyright
      -notice.txt'
externalDocs:
  description: ETSI MEC013 V1.1.1 Location Service API
  url: 'http://www.etsi.org/deliver/etsi_gs/MEC
    /001_099/013/01.01.01_60/gs_mec013v010101p.pdf'
host: '127.0.0.1:8081'
basePath: /exampleAPI/location/v1
schemes:
  - http
  - https
consumes:
  - application/json
produces:
  - application/json
tags:
  - name: zones
  - name: users
  - name: subscriptions
parameters:
  Body.UserTrackingSubscription:
    name: userTrackingSubscription
    in: body
    description: User Tracking Subscription
    required: true
    schema:
      $ref: '#/definitions/UserTrackingSubscription'
  Body.ZonalTrafficSubscription:
    name: zonalTrafficSubscription
```

Location API 1.1.1

[Base URL: 127.0.0.1:8081/exampleAPI/location/v1]
https://forge.etsi.org/gitlab/mec/gs013-location-api/raw/master/LocationAPI.yaml

The ETSI MEC ISG MEC012 Location API described using OpenAPI. The API is based on the Open Mobile Alliance's specification RESTful Network API for Zonal Presence

ETSI Forge copyright notice

ETSI MEC013 V1.1.1 Location Service API

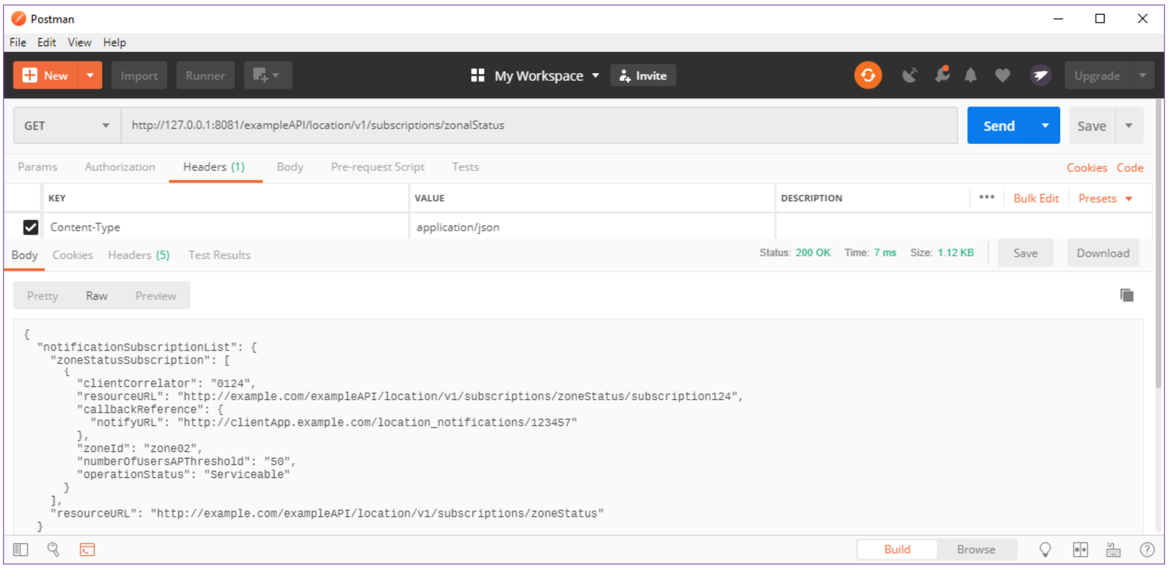
Schemes: HTTP

zones

- GET /zones
- GET /zones/{zoneId}
- GET /zones/{zoneId}/accessPoints

Figure 9: MEC Location API presented using the Swagger Editor

To interact with the server from a client perspective (service consuming MEC application) third party tools such as postman (https://www.getpostman.com/) can be used as an initial step to test out the offered requests and see example responses, see Figure 10. Such tools are rather more user friendly than the Client URL (cURL) command-line tool for transferring data using URL syntax. However, the Swagger Editor does provide the cURL syntax for querying each request endpoint of the API. This is seen, for instance, by clicking GET /zones (Figure 9) and then clicking the “Try it out” button that will appear.



GET http://127.0.0.1:8081/exampleAPI/location/v1/subscriptions/zonalStatus

	KEY	VALUE	DESCRIPTION
☑	Content-Type	application/json	

Status: 200 OK Time: 7 ms Size: 1.12 KB

```
{
  "notificationSubscriptionList": {
    "zoneStatusSubscription": [
      {
        "clientCorrelator": "0124",
        "resourceURL": "http://example.com/exampleAPI/location/v1/subscriptions/zoneStatus/subscription124",
        "callbackReference": {
          "notifyURL": "http://clientApp.example.com/location_notifications/123457"
        },
        "zoneId": "zone02",
        "numberOfUsersAPThreshold": "50",
        "operationStatus": "Serviceable"
      }
    ],
    "resourceURL": "http://example.com/exampleAPI/location/v1/subscriptions/zoneStatus"
  }
```

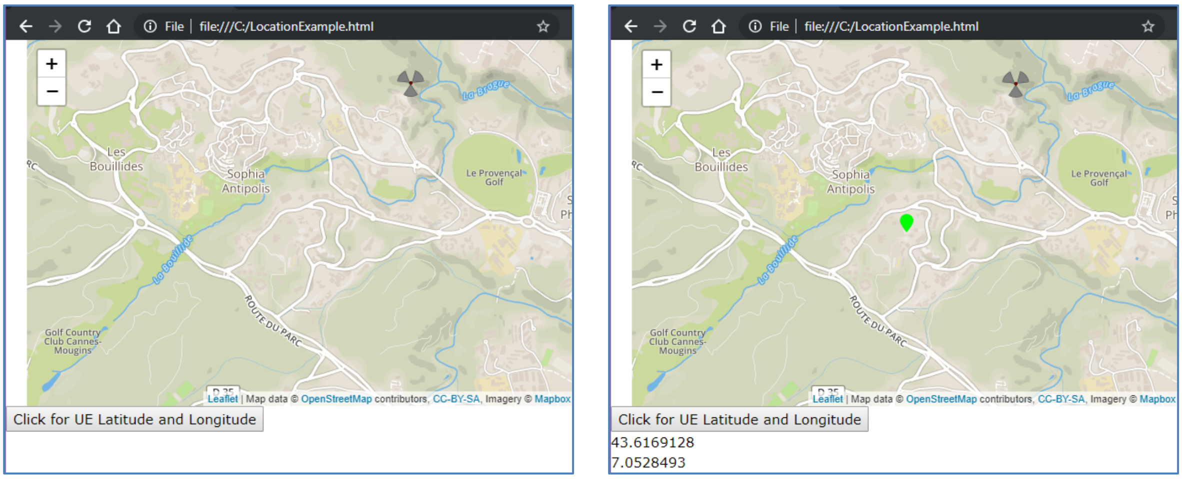
Figure 10: Example MEC Location API query and response

Third party libraries have also been developed to interact with interface files directly, thereby removing the server dependency. For example, Swagger-JS (https://github.com/swagger-api/swagger-js), which just requires a URI link to the interface file. The interface file maybe hosted by the local server, but it is just the file that is required and therefore the version hosted on Forge is equally applicable (e.g. https://forge.etsi.org/gitlab/mec/gs013-location-api/raw/master/LocationAPI.json in the case of the MEC Location API). The JavaScript file can, for example, be included in a HTML file acting as a MEC application:

```
<script src='swagger-client.js' type='text/javascript'></script>
```

This then offers the capability for the various API methods to be called from with the HTML file, where any API requests are validated against the interface file and the associated responses are populated with the example data from that file. This approach has been used in generating the plots in Figure 11 using the Leaflet JavaScript library (https://leafletjs.com/) for the interactive maps.

Finally, in order to generate a client that interacts with a server, “Generate Client” can be selected from the Swagger Editor (Figure 9). As with server generation, many different language options are supported. Once developed, MEC Applications can be packaged, on-boarded and then instantiated according to the phases detailed in another section. This is in the knowledge that the application will be compliant to the MEC API specifications, since its development was based on the client stub produced from those specifications through the OpenAPI compliant interface description file.


File | file:///C:/LocationExample.html
+
−
Les Bouillides
Sophia Antipolis
Le Provençal Golf
La Brague
La Bouillide
ROUTE DU PARC
Golf Country Club Cannes-Mougins
Leaflet | Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © Mapbox
Click for UE Latitude and Longitude

```
43.6169128
7.0528493
```

Figure 11: Plotting user location based on calling the GET users by ID via the MEC Location API

Other aspects relevant for developers

Implementation aspects

Different level of Hardware and Software Security mitigations/measures are recommended. For example, for hardware, the environment could use a Hardware Security Module (HSM) for certificates. A Trusted Platform Module (TPM) and trusted boot could also be used. For software, communication internally and externally could be secured with Transport Layer Security (TLS). In most cases, only authorized traffic is allowed in the MEC.

The MEC also must prevent illegal access from dishonest terminals and dishonest MEC application developers. The MEC system must secure the environment for running services for the following actors:

- User
- Network operator
- Third-party application provider
- Application developer
- Content provider
- Platform vendor

The Standard 4G/5G packet core protocols protect from unauthorized users joining the network. Access Control Lists (ACLs) can control policies that are leveraged by sessions and applications running at the Edge. MEC services are configurable for DeMilitarized Zone (DMZ), OAM management, and LTE network.

Security

With both economic and political cyber-threats on the rise and sophistication of attacks increasing, security considerations should be front and centre in designing a MEC application. In many ways, MEC-enabled applications are like other modern applications in terms of security. Approaches, based on modern concepts, such as “zero-trust networking” should be used. As we noted, a microservices-based design approach (when implemented well) lends itself to isolation of vulnerabilities.

However, MEC can also present unique security challenges as well as opportunities when it comes to the specifics of edge computing. To illustrate this, we provide some examples:

- Challenges:
    - Each MEC “mini-cloud” has limited compute capabilities and these are likely to be costlier. Thus, MEC mini-clouds may be more susceptible to DoS (Denial of Service) attacks than more central clouds.
    - Physical security must be a real consideration with edge computing even for application developers. An application developer is not likely to worry about what happens if a malicious party accesses the physical infrastructure of a traditional cloud provider – this is an extremely unlikely event. However, this is not necessarily true for MEC – edge compute clusters are often located in much less physically secure locations.
    - New security attack surface due to more entry points into application, more complicated certificate management.

  - MEC Services may contain sensitive information. Therefore, every request for information within the MEC host must be authenticated and authorized.
  - As different MEC application vendors may be installed side by side on the same MEC infrastructure, data separation policies must be taken in account carefully.
- Opportunities:
  - The collection of edge and cloud taken together presents a much more challenging attack surface than a centralized cloud (even an internally redundant and distributed one). As such, MEC may present an interesting solution for safeguarding critical data and compute tasks that otherwise do no need to be run at the edge.
  - While more vulnerable to a DoS attack, the limited reachability of many MEC PoPs may make it significantly more difficult to perpetrate a DDoS attack against them.
  - Knowledge of subscriber by network, location awareness can be used to strengthen authentication process, introduce new types of heuristics in fraud detection, availability of Edge-focused application firewalls
  - Since the MEC host location is much closer to the access network than the core network, every network blocking resulting from attack will have a much smaller impact on the overall network. Using MEC, wide scale server connectivity shutdown will be avoided.

In summary, designing applications for the edge requires careful consideration of security. MEC can be used to improve the overall security of an application – but if not used properly, it can also expose the application to new or increased threats.

Mobility

The terminal is likely to be a mobile device and the current MEC host of the user session may not be the best choice due to a change in the device's location. The MEC system allows the relocation of the user context for a session from one application instance to another running in a MEC host closer to the user. This facilitates service continuity and offers programmers the opportunity to design their applications with the capability to leverage and optimize the user context relocation procedure. Such capabilities include monitoring the application KPIs to assist the relocation decision, mechanisms to determine the right relocation timing and data synchronization. Details of what constitutes user context are provided later in this section.

Application mobility may be triggered by a change in the device bearer path in the underlying transport network. For example, device handover from one 4G or 5G cell to another associated with a different MEC host. This situation requires the MEC system to make the application instance available in this new MEC host to ensure the optimum lowest latency service to this device. If a user context existed in the source application instance, it must be transferred to the new target application instance.

Support by the MEC system for application mobility is aimed at optimizing the performance of applications and reducing service latency by relocating user sessions to the application instance closest to the end user. This facilitated through the MEC application mobility service API. This service allows applications to manage the relocation of user sessions between application instances by informing the source instance of the target instance and by enabling user context transfer between the two instances.

The entities involved in application mobility are highlighted in Figure 12. Here, the client app runs on a device such as a mobile phone or on equipment installed in vehicle. The device hosting the client app can communicate via cellular network with the MEC hosts, i.e. MEC Host (A) and (B). These hosts are where the MEC app’ and app’’ instances run. At the remote level, the application is implemented in the cloud backend to serve the client app. The client app can communicate with either the edge level MEC application instances or the application instance(s) in cloud backend via the underlying network to receive the service produced by the application.

Initially, the client app may communicate with the remote level application. Then when the MEC system identifies that the device is within the serving area of MEC Host (B) it may create an edge level application instance (e.g. MEC app’’) to serve the user and offer an enhanced service to the client app, e.g. contextual based on the device location with overall lower latency. If the MEC system identifies device handover to a new location associated with MEC Host (A) through the application mobility service, it will then create another new application instance (i.e. MEC app’) to continue serving the user. The application may need to synchronize the user context between application instances if they are running on the cloud backend, MEC Host (A) or (B) to ensure that the application service continuity is maintained.

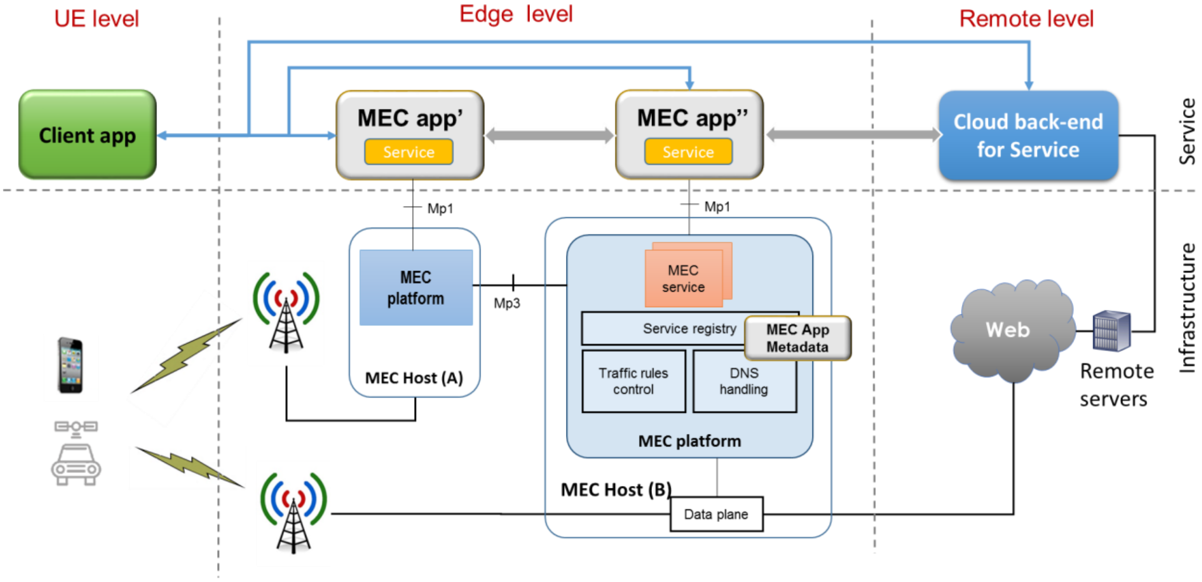


Figure 12: Application mobility support transferring the service between the Cloud back-end and MEC instance

User context is the application-specific runtime data maintained by the MEC application that is associated with a specific user or a group of users being serviced by that application. It may contain the client application identifier, the status of the application instance serving the user, the communication link and other runtime information. Such information is specific to and belongs only to the application. User context synchronization depends on the application implementation:

- For a stateless application, i.e. an application that does not retain the service state or recorded data about the user for use in the next service session, the application can be relocated to another MEC host without using the mobility service.
- For a stateful application, i.e. an application that stores information about service state during an application session change, it will require further application logic to transfer the user context from the source to the target application instance to maintain the service continuity.

MEC application mobility service can facilitate the support of service continuity for stateful applications, specifically by providing the necessary communication connections and support information exchange between application instances. However, the synchronization of application service state and other specific runtime information relies on the application SW implementation, which could take one of two approaches:

- Store it in the client app of user device
- Store it in the MEC application instance

If the user context is stored in the client app, the client app may be required to send the user context to the target application instance after the application service is moved to another MEC host.

If the user context is stored in the MEC application, the application is required to support the capability to transfer the user context between MEC application instances. In this case the MEC application mobility service API can assist since it allows the application to subscribe to notifications on relevant mobility events. This enables the application to prepare the user context for transfer to the target instance in a timely manner. The mobility service can also be used to notify the source application instance of the address of its peer target instance and therefore where the user context needs to be transferred.

Future evolutions: Edgy DevOps

In the previous sections, it has been highlighted how the microservices variant of the service-oriented architecture (SOA) pattern has direct applicability to the software model envisaged for MEC and its services and applications. The approach's ability to compose a distributed application from separately deployable services was also described. Such services are expected to communicate via web interfaces (such as MEC's RESTful API approach) and perform a specific business function. In combination with this approach DevOps practices are considered as being highly complementary.

Currently the ecosystem is quickly moving to use Function as a Service (FaaS) approaches like Greengrass for Amazon's AWS Lambda, Microsoft's Azure IoT stack and GE's Predix to enable it. Also, DevOps seems to be a greenfield for MEC, as it is considered to be a business-driven software delivery approach. DevOps is based on lean and agile principles in which cross-functional teams collaborate to deliver software in a continuous manner. Such teams consist of representatives from all disciplines responsible for developing and deploying software services, including the business owners, developers, operations and quality assurance. This continuous delivery (CD) based approach enhances a business' ability to exploit new market opportunities and quickly adapt to customer feedback. The MEC ecosystem is evolving and presents opportunities for the development of new and innovative services, which means that the DevOps approach is ideally suited and offers a genuine path to deriving new business value.

Adopting the microservices approach results in services that are modularized and small in nature. Given their size, each individual service is easier to develop, test and maintain. The services may also be developed and deployed in parallel. This facilitates continuous integration (CI) and delivery, which are key principles of the DevOps approach. DevOps teams are also well placed to take the individual pieces of functionality offered through microservices and use those as the building blocks for larger applications and systems. Those systems can then be easily expanded by adding new microservices without unnecessarily affecting other parts of the application, thereby offering flexibility and achieving scalability. The consequence is that software development organizations already employing DevOps practices to develop microservices will already be well placed to embrace the MEC software development paradigm, where an overall MEC solution will comprise of multiple software components (i.e. microservices) each providing different capabilities and functions, potentially from different providers, and where those components are expected to continue to expand and evolve thereby benefiting from DevOps CI and CD processes. This expansion will happen at a high and low level, for instance at a high level as the MEC platform's overall capabilities are enhanced to support multiple access technologies and full application mobility and at a lower level as specific applications are further developed to exploit enhanced API functionality, for instance as the Radio Network Information Service capabilities expand into the multi-access domain.

An activity closely related to implementing the DevOps approach is the efforts underway in the MEC ISG to define a test framework to cover aspects such as conformance and interoperability. The framework will deliver a test suite that is ideal for automated testing. This is considered an integral DevOps component, since automated testing supports the delivery pipeline in creating processes that are iterative, frequent, repeatable, and reliable.

Figure 13 shows an overview of different cloud business models where different key touch points are expected to be required and standardized for the different cases.

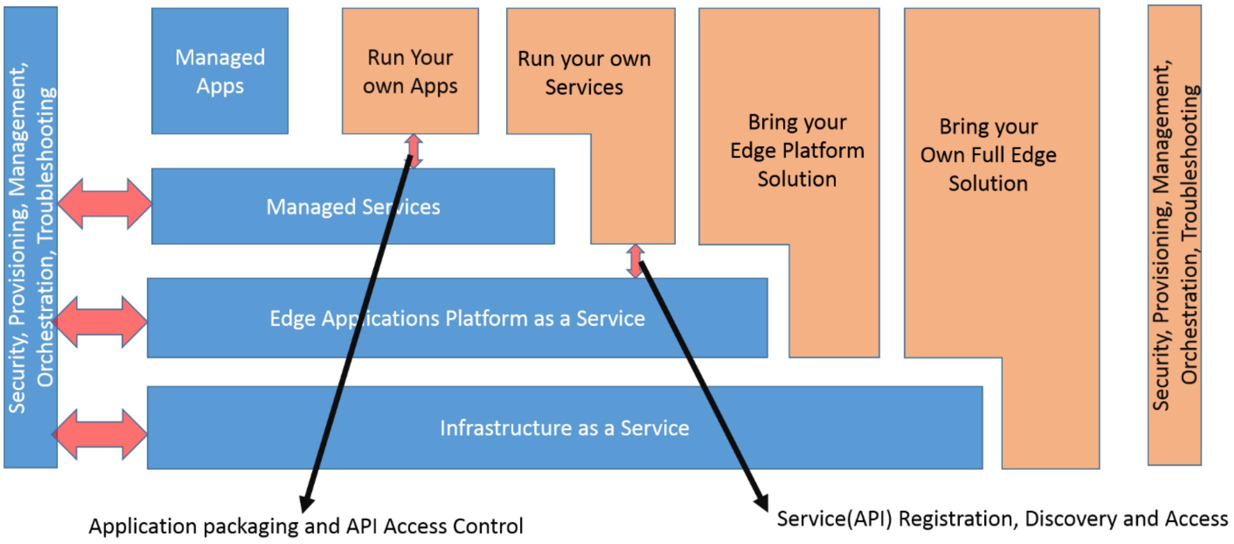

Security, Provisioning, Management, Orchestration, Troubleshooting
Managed Apps
Run Your own Apps
Run your own Services
Bring your Edge Platform Solution
Bring your Own Full Edge Solution
Security, Provisioning, Management, Orchestration, Troubleshooting
Managed Services
Edge Applications Platform as a Service
Infrastructure as a Service
Application packaging and API Access Control
Service(API) Registration, Discovery and Access

Figure 13: Overview of different cloud business models

Based on this layered model, while the fundamental principles of DevOps such as Continuous Integration and Continuous Delivery (CI/CD) apply across the board, the realization of such practices will differ significantly depending on the software layer where the development is performed and the management boundaries of the Edge Stack. Developers of enterprise applications or operator centric Network Functions will have a different deployment environment compared to consumer application developers and the tools used to deploy an application to a CPE will be different than for one deployed in an Edge Data Centre. One of the key operational requirements for Edge Computing is also the concept of Zero-Touch provisioning. This applies to all the layers of the edge stack and requires full management automation and service assurance.

To address these variations in the broad scope for edge computing for telco and enterprise applications, Akraino a relatively new Linux Foundation project attempts to break up the end to end framework into functional modules and use case driven blueprints (integrated stacks). This is an integration project in the sense that most of the code is being integrated from upstream projects via CI/CD tools. The resulting Edge Stack deliverables are tested, deployable platforms for telco and enterprise edge applications. Akraino Edge Stack brings enterprise application developers cloud deployment and management tools tailored for the edge application development.

Consumer application developers will look at a more abstract deployment model where the application follows an intent based, declarative approach that decouples the functions from the platform and execution environment. In this scenario, much of the DevOps process is being pushed to the platform including application package validation, testing, dependency checking, app update or roll back. To achieve this, the application must adhere to strict packaging specifications including well defined manifests, descriptors and operational models for scaling/support.

There are many requirements that make automation mandatory for the Edge Computing ecosystem to enable Zero Touch provisioning and while this improves the overall integration, DevOps is not eliminated. It just gets shifted into the platform at much higher layers in the stack.

FaaS make development of applications easier with less code to write and with a clear separation of concern between the function implementation and its consumers.

While application developers can and should be able to compose applications out of available functions using less code it does not eliminate the need to address compatibilities of their app with Function/Microservice versions deployed and consequentially deal with updates/upgrades of the platform.

Concluding Remarks

This paper addressed the issues that application developers face when developing applications for edge computing, including MEC-based edge computing. While offering only a superficial treatment, the paper provided a guide to developers for further reading and a starting point on how to address the challenges posed by this new type of application hosting environment.

All the authors of this white paper are active in ETSI ISG MEC. As the only international standards committee focused on edge computing, ETSI ISG MEC continues to work on simplifying the application development process and enabling interoperability through the definition of a reference architecture, standardization of APIs and other activities in this space. We invite the readers to learn more by visiting our web pages, http://www.etsi.org/mec. For those interested in participating in our work, the web page also contains a link to information on how to join our group.

Annex A - useful material for SW developers

This annex contains a list of useful material for software developers for MEC, e.g. references to repositories, open source communities and software frameworks.

In addition, few examples of collaborative projects related to MEC are provided.

Name	description	Link
ETSI MEC specifications	Hub with the latest published MEC specifications.	https://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing
MEC API definitions	GIT Repository with an OpenAPI compliant implementation of MEC APIs	https://forge.etsi.org/
ETSI NFV specs	Hub with latest Network Functions Virtualisation specifications	https://www.etsi.org/technologies-clusters/technologies/nfv
NEV SDK	NFV platform for MEC applications and services	https://networkbuilders.intel.com/network-technologies/nev
OpenAPI specification	OpenAPI GitHub	https://github.com/OAI/OpenAPI-Specification
OpenAPI Initiative	OAI Developer Community	https://www.openapis.org
Data Plane Development Kit (DPDK)	Library to accelerate packet processing	https://www.dpdk.org/
MEC wiki	ETSI wiki for MEC	https://mecwiki.etsi.org
Openstack	Open Infrastructure website and introduction to MEC	https://www.openstack.org/edge-computing/cloud-edge-computing-beyond-the-data-center/
Development resources for OpenStack clouds	Resources for application development on OpenStack	https://developer.openstack.org/ https://www.openstack.org/edge-computing/

Annex B - Collaborative projects related to MEC

Name **5GMedia (H2020)** http://www.5gmedia.eu/

Description 5G-MEDIA aims at delivering an integrated programmable service platform for the development, design and operations of media applications in 5G networks by providing mechanisms to flexibly adapt service operations to dynamic conditions and react upon events (e.g. to transparently accommodate auto-scaling of resources, VNF re-placement, etc.) by using an ETSI MANO framework.

The role of ETSI MEC is relevant to manage and extend the core cloud-computing capabilities towards the edge of the mobile network, within the Radio Access Network (RAN) and in close proximity to mobile users. 5GMedia aims at demonstrating a variety of media related use cases, such as the dynamic instantiation and management of media caches to serve Ultra High Definition contents to moving mobile devices.

Name **5GCity (H2020)** https://www.5gcity.eu/

Description The 5GCity project is working to design, implement and deploy a distributed cloud, edge and radio platform for smart cities and infrastructure owners acting as 5G Neutral Hosts. In 5GCity, cloud and edge computing technologies are integrated and extended to address 5G requirements and implement network slicing and end to end service orchestration from the radio up to the core data centres. To validate the 5GCity platform we are developing challenging media-related use cases which will be soon deployed in live 5GCity infrastructures in the cities of Barcelona (ES), Bristol (UK) and Lucca (IT).

The 5GCity platform is designed to leverage both ETSI MEC and ETSI NFV architectures and interfaces to implement the neutral hosting features. We have followed the guidelines provided by ETSI MEC (ETSI GR MEC 017 V1.1.1) and designed a concrete integration of ETSI MEC and NFV through a thin orchestration layer on top of the NFV and MEC orchestrators. This new orchestrator enables the coexistence of the different descriptors used in the two frameworks and a smooth communication between them across the different layers and scopes of action. In 5GCity, the MEC orchestrator controls the edge applications and the edge platform management, while the NFV orchestrator performs the actual functions deployment on the NFV Infrastructure (NFVI). The NFV orchestrator is aware of the deployment of MEC/Edge Apps which is achieved by their deployment as VNFs (as recommended by ETSI MEC specifications.

The 5GCity consortium is formed by 18 partners from 7 European countries, including leading vendors, network and infrastructure operators, research centres and SMEs all heavily involved in 5G, MEC and NFV development. The 5GCity project is a 5G PPP Phase 2 project, funded by the European Commission under the of the Horizon 2020 programme.

Name **NRG-5 (H2020)** www.nrg5.eu

Description The NRG-5 project envisages contributing to the 5G PPP/5G Initiative research and development activities and participation at the relevant 5G Working Groups by delivering a novel 5G-PPP compliant, decentralized, secure and resilient framework, with highly availability able to homogeneously model and virtualize multi-homed, static or moving, hardware constrained (smart energy) devices, edge computing resources and elastic virtualized services over electricity and gas infrastructure assets combined with the telecommunications infrastructure covering the full spectrum of the communication and computational needs.

The ultimate project goal is to render the deployment, operation and management of existing and new communications and energy infrastructures (in the context of the Smart Energy-as-a-Service) easier, safer, more secure and resilient from an operational and financial point of view.

Name **5GEssence** http://www.5g-essence-h2020.eu/

Description 5G ESSENCE addresses the paradigms of Edge Cloud computing and Small Cell as a Service by fuelling the drivers and removing the barriers in the Small Cell market, forecasted to grow at an impressive pace up to 2020 and beyond and to play a key role in the 5G ecosystem. 5G ESSENCE provides a highly flexible and scalable platform, able to support new business models and revenue streams by creating a neutral host market and reducing operational costs by providing new opportunities for ownership, deployment, operation and amortization.

The technical approach exploits the benefits of the centralization of Small Cell functions as scale grows through an edge cloud environment based on a two-tier architecture: a first distributed tier for providing low latency services and a second centralized tier for providing high processing power for computing-intensive network applications. This allows decoupling the control and user planes of the Radio Access Network (RAN) and achieving the benefits of Cloud-RAN without the enormous fronthaul latency restrictions. The use of end-to-end network slicing mechanisms will allow sharing the 5G ESSENCE infrastructure among multiple operators/vertical industries and customizing its capabilities on a per-tenant basis. The versatility of the architecture is enhanced by high-performance virtualization techniques for data isolation, latency reduction and resource efficiency, and by orchestrating lightweight virtual resources enabling efficient Virtualized Network Function placement and live migration.

Name **Matilda (H2020)** http://www.matilda-5g.eu/

Description The vision of MATILDA is to design and implement a holistic 5G end-to-end services operational framework tackling the lifecycle of design, development and orchestration of 5G-ready applications and 5G network services over programmable infrastructure, following a unified programmability model and a set of control abstractions.

It aims to devise and realize a radical shift in the development of software for 5G-ready applications as well as virtual and physical network functions and network services, through the adoption of a unified programmability model, the definition of proper abstractions and the creation of an open development environment that may be used by application as well as network functions developers.

Intelligent and unified orchestration mechanisms will be applied for the automated placement of the 5G-ready applications and the creation and maintenance of the required network slices. Deployment and runtime policies enforcement is provided through a set of optimisation mechanisms providing deployment plans based on high level objectives and a set of mechanisms supporting runtime adaptation of the application components and/or network functions based on policies defined on behalf of a services provider.

Multi-site management of the cloud/edge computing and IoT resources is supported by a multi-site virtualized infrastructure manager, while the lifecycle management of the supported Virtual Network Functions Forwarding Graphs (VNF-FGs) as well as a set of network management activities are provided by a multi-site NFV Orchestrator (NFVO). Network and application-oriented analytics and profiling mechanisms are supported based on real-time as well as a posteriori processing of the collected data from a set of monitoring streams. The developed 5G-ready application components, applications, virtual network functions and application-aware network services are made available for open-source or commercial purposes, re-use and extension through a 5G marketplace.

Name **5G-Coral** http://5g-coral.eu/

Description 5G-CORAL project leverages on the pervasiveness of edge and fog computing in the Radio Access Network (RAN) to create a unique opportunity for access convergence. It envisions a distributed edge computing platform also suitable for constrained devices.

Name **5G-Transformer** http://5g-transformer.eu

Description 5G-Transformer aims to transform today's rigid mobile transport networks into an SDN/NFV-based mobile transport and computing platform. It envisions an edge computing platform capable of offering services tailored to the specific needs of vertical industries.

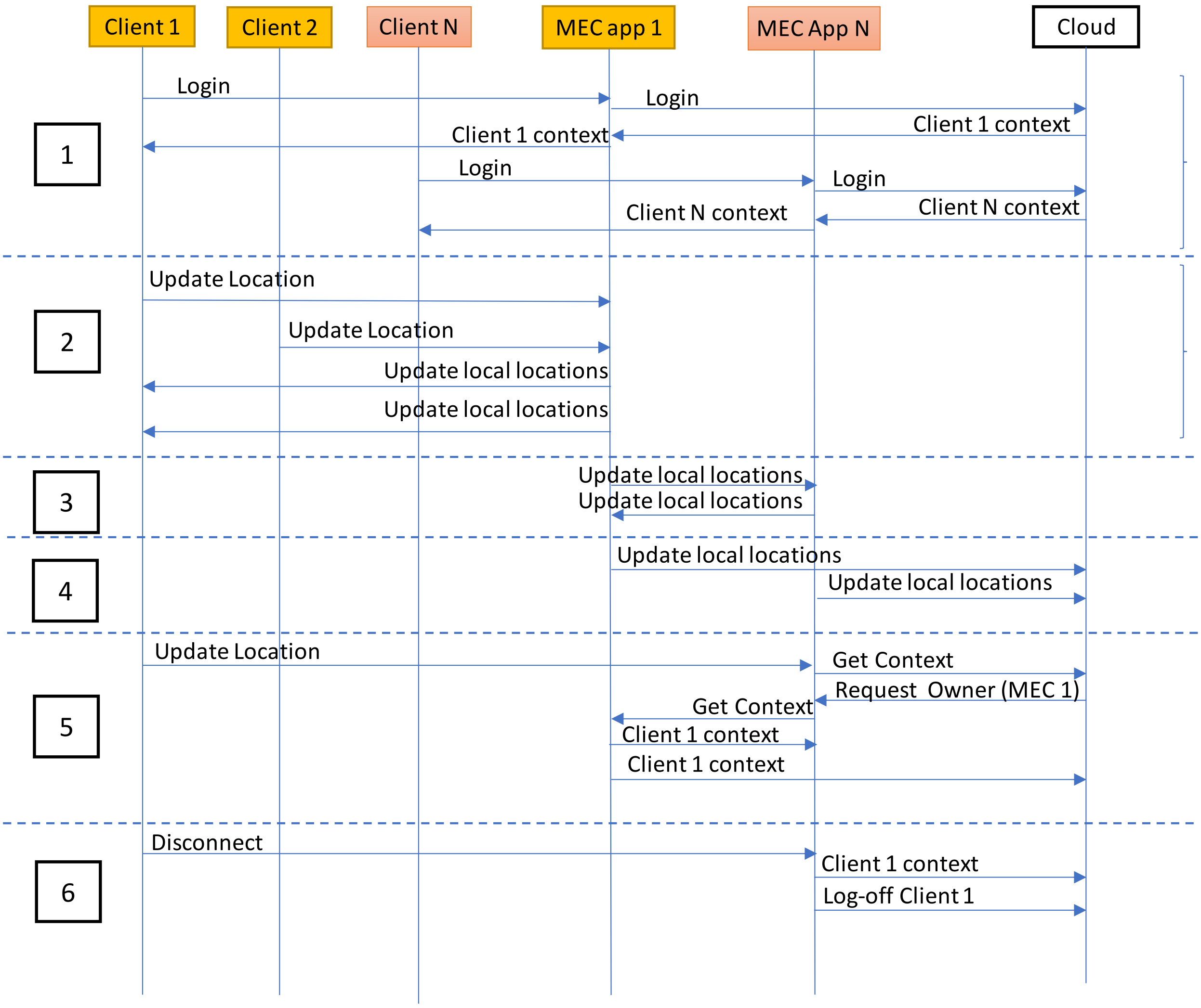
Annex C - Exemplary use cases for developers

V2X service on collision prevention

This use case illustrates a possible implementation of collision prevention V2X service, where all client apps running on vehicles regularly share their locations, directions and speed with others in proximity, so each client can detect potential collision and make preventive actions like change direction or speed.

This use case is very latency critical, where delayed decisions could be the difference between whether a collision happens or not. For instance, at 60 km/h a vehicle moves by 16 cm every 10 ms.

In the example below clients 1 and 2 are served by MEC instance 1 and client N by MEC app instance N. Client 2 is already connected, but client 1 has to initiate connection.



1. Client 1 connects to local MEC app instance 1. It establishes TCP connection to MEC Application using DNS redirection by MEC platform to resolve known domain name. Once connection is established, it uses Login API and provides credentials. MEC app 1 can't authenticate the user locally and therefore forwards the request to Cloud back-end, which verifies credentials and

responds with a Single Sign-On token and Client 1 Context. The same process takes place with Client N and MEC app N instance.

2. Service Apps running on clients regularly shares location, time of measurement, direction and speed with MEC instances. MEC app accumulates the locations of served Clients and updates each locally served Client with recent updates containing locations of clients in close-proximity (e.g. within a 100m/300ft radius).
    - Each client computes the possibility of collision using its current location and the extrapolated location of nearby clients.
3. MEC regularly sends the list of Clients which are in proximity of areas served by other MEC app instances to those instances (which is important for smooth hand-overs). Prior to this step, MEC apps discover peers and their physical location with assistance of MEC platform.
4. MEC instances updates the cloud-backend with latest information about clients on a regular basis, though significantly less frequently than updates to MEC app peers and clients.
5. Client 1 moved to area served by MEC app instance N and its traffic is now being handled by that instance. MEC Instance N doesn't yet know this client but receives its "single sign on" (SSO) token and forwards the request to Cloud backend to validate it and get associated context. The cloud back-end maintains a directory of which client is served by which MEC app instance and refers to MEC 1 to request context. MEC N sends request to MEC 1 which responds with context to both MEC N and Cloud and drops local context as it's now owned by MEC N.
6. Once client 1 disconnects, the MEC instance updates the cloud with the recent context and sends a Logoff message to drop the authentication token and Client's 1 context.

Video transcoding use case

With the advent of 5G, the media sector is witnessing the emergence of a number of innovative services. Among these, advanced video applications are attracting a particular attention. Many factors can explain such interest. According to Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016-2021, White Paper [8], video data traffic will grow to 82% of all internet traffic by 2021. Moreover, consumers' preferences have shown a clear shift from traditional services towards advanced video activities and social networking, carried out through a variety of personal devices, such as smartphones, tablets, smart watches, personal computers, etc.

During sport events or concerts, consumers increasingly often wish to originate and share high quality media contents in groups of peers, besides merely receiving them from a centralized distribution system. The produced contents must be available as flash events – i.e. in the shortest possible time - and adapted to the consumers' device capabilities, to optimize the perceived Quality of Experience.

In touristic locations or relevant cultural sites, the use of ad hoc multimedia contents, 3D reconstructions (of monuments, statues, or buildings etc.), and panoramic videos can be combined with geo-localization services, augmented reality, and object recognition capabilities, to offer highly immersive experiences to end users. For instance, in a historical location, tourists could see on their personal devices the reconstruction of an antique building overlapping the real landscape.

Using the architecture depicted in another section, Figure 2, edge applications can be developed and used to provide an immersive experience to end users. To do so several different interacting components are required, concurrently running on the end users' devices, in the virtualization infrastructure at the edge, and in one or more core data centres.

Consumers can access the immersive video services through a specific application, downloadable on their personal device, or, alternatively, through any browser. This way, they can register to the service and create groups of peers sharing media contents, off-line or in real-time. The end user app also provides geo-localization data to the MEC App components running at the edge and sends and receives the multimedia streams that originate the immersive experience.

The MEC App running at the edge consists of three different components (that can run both as Virtual Machines and containers), namely the dataBase (dB), the eStore and the Media Processing Unit (MPU) components. Such components are independent entities, which offer their services through Restful APIs registered on the MEC platform.

The dB component collects and monitors relevant information about users and groups. When consumers connect to the local access network for the first time, they can download the end user app. The next step is the registration to the system, which identifies each user by a name and/or nickname. Through the interaction between end-user app and the dB component, consumers create groups that can share media contents. For each group member, the dB component saves the relevant information about identity, group belonging, connectivity, and Service Level Agreement, to provide the required type of service. Moreover, it provides basic security functionalities, such as user password management, and records the geo-localization data periodically sent by the end-user device app.

The eSTORE component offers local storage capabilities to the groups of users. This way, the operation of content upload and/or download to the storage system can be significantly speeded-up, since it does not require any access to the core network. This feature is particularly relevant during crowded events, when the backhaul connection becomes a contended resource that can originate bottlenecks for any activity involving the core network.

The basic task of the MPU component is converting audio/video streams from one format to another, to adapt to the capabilities of the receiving end-user device. A multimedia content could reside as a pre-recorded file in the storage system or come as a real-time packet stream. The requested transcoding service can be mono-directional, as in video stream distribution applications, or bi-directional, as in real-time videoconferencing.

The MPU supports the most popular video codecs, such as H.265 and Google VP9. It can take advantage of the potential presence of GPUs at the edge to improve overall performance and efficiency. Video streams originated by consumers are saved in the distributed storage system (eSTORE), to be shared later. Pre-recorded contents, for instance coming from a broadcaster archive, can be streamed on demand to single users, to an entire group, or to all the users.

The MPU offers the advanced audio/image/video processing capabilities needed to provide immersive video services. Specifically, it can stream to consumers 360° video data originated from specific multi-sensor cameras. To this end, the MPU processes the different video contributions coming from the camera, performing all the needed adjustments in terms of image stitching, colour correction, projection in the equi-rectangular format, etc, - so that consumers can select in real-time on their device the viewing direction in the observed scene.

To enhance the immersive experience, the MPU can provide 3D reconstructions of spaces (such as a square or a view) or objects (monuments, building etc.). In addition, it can recognize objects in the images and video streams sent by end users, to perform automatically visual searches in ad hoc image or video databases. The visual search allows matching images or videos captured by the user, such as buildings, statues, paintings, with contents present in a database, exploiting visual similarities and without the need for manual query.

Finally, certain specific components of the overall immersive video application can run in the core cloud. For instance, they allow the permanent storage of multimedia contents originated during a crowded event, temporarily saved at the network edge. In addition, specific back-end components can run in the core cloud, to process large data sets during the training phase of machine learning algorithms, or to run the most computationally intensive and time-consuming activities in 3D reconstruction.

The MEC framework can significantly contribute to the successful implementation and provision of the immersive video services above described. To this end, it can offer not only low latency, local computing and storage resources, and high data-rates, but also localization awareness and flexibility to radio link conditions.

From the perspective of MEC App developers, the capabilities offered by the MEC Platform and the MEC management blocks can dramatically simplify both the development phase of new applications, and the entire lifecycle of the offered services.

As an example, the coordinated activities of the MEC management blocks and the MEC Platform can straightforwardly solve in a fully automated way the complex problem of properly steering the user data towards the corresponding MEC App running in the local edge infrastructure. To this end, App developers must only specify, during the on-boarding process of an MEC App, a Fully Qualified Domain Name, which identifies the implemented service; the overall MEC framework has the capability to produce and enforce into the MEC data plane the corresponding DNS and traffic steering rules.

References

[1] Mobile-Edge Computing – Introductory Technical White Paper, September 2014; https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf

[2] ETSI White Paper No. 11: "Mobile Edge Computing: A key technology towards 5G", ISBN No. 979-10-92620-08-5, first edition, September 2015; https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf

[3] ETSI GS MEC 011: "Mobile Edge Platform Application Enablement", http://www.etsi.org/deliver/etsi_gs/MEC/001_099/011/01.01.01_60/gs_MEC011v010101p.pdf

[4] ETSI GS MEC 016: "UE application interface", http://www.etsi.org/deliver/etsi_gs/MEC/001_099/016/01.01.01_60/gs_MEC016v010101p.pdf

[5] ETSI GS MEC 010-2: "Mobile Edge Management; Part 2: Application lifecycle, rules and requirements management", http://www.etsi.org/deliver/etsi_gs/MEC/001_099/01002/01.01.01_60/gs_MEC01002v010101p.pdf

[6] IETF RFC 6750: "The OAuth 2.0 Authorization Framework: Bearer Token Usage", https://tools.ietf.org/html/rfc6750

[7] https://restfulapi.net/

[8] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper, Updated:March 28, 2017; https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html

[9] ETSI GS MEC 012: "Mobile Edge Computing (MEC); Radio Network Information API", V1.1.1, July 2017. https://www.etsi.org/deliver/etsi_gs/MEC/001_099/012/01.01.01_60/gs_MEC012v010101p.pdf

[10] ETSI GS MEC 013: "Mobile Edge Computing (MEC); Location API", V1.1.1, July 2017. https://www.etsi.org/deliver/etsi_gs/MEC/001_099/013/01.01.01_60/gs_MEC013v010101p.pdf

[11] ETSI GS MEC 009: "Multi-access Edge Computing (MEC); General principles for MEC Service APIs", V2.1.1, January 2019. https://www.etsi.org/deliver/etsi_gs/MEC/001_099/009/02.01.01_60/gs_MEC009v020101p.pdf

[12] ETSI GS MEC 006: "Mobile Edge Computing (MEC); Market Acceleration; MEC Metrics Best Practice and Guidelines", V1.1.1, January 2017. https://www.etsi.org/deliver/etsi_gs/MEC-IEG/001_099/006/01.01.01_60/gs_MEC-IEG006v010101p.pdf

[13] ETSI GS MEC 003: "Multi-access Edge Computing (MEC); Framework and Reference Architecture", V2.1.1, January 2019. https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf

[14] ETSI GS MEC 015: "Mobile Edge Computing (MEC); Bandwidth Management API", V1.1.1, October 2017. https://www.etsi.org/deliver/etsi_gs/MEC/001_099/015/01.01.01_60/gs_MEC015v010101p.pdf

[15] ETSI GS MEC 014: "Mobile Edge Computing (MEC); UE Identity API", V1.1.1, February 2018. https://www.etsi.org/deliver/etsi_gs/MEC/001_099/014/01.01.01_60/gs_MEC014v010101p.pdf

[16] SCF 084.07.01: "Small cell zone services - RESTful bindings".

[17] SCF 152.07.01: "Small cell services API".

ETSI
06921 Sophia Antipolis CEDEX, France
Tel +33 4 92 94 42 00
info@etsi.org
www.etsi.org

This White Paper is issued for information only. It does not constitute an official or agreed position of ETSI, nor of its Members. The views expressed are entirely those of the author(s).

ETSI declines all responsibility for any errors and any loss or damage resulting from use of the contents of this White Paper.

ETSI also declines responsibility for any infringement of any third party's Intellectual Property Rights (IPR), but will be pleased to acknowledge any IPR and correct any infringement of which it is advised.