ETSI White Paper No. 20

Developing Software for Multi-Access Edge Computing

First edition – September 2017

ISBN No. 979-10-92620-14-6

Authors:

Alex Reznik (editor), Rohit Arora, Mark Cannon, Luca Cominardi, Walter Featherstone, Rui Frazao, Fabio Giust, Sami Kekki, Alice Li, Dario Sabella, Charles Turyagyenda, Zhou Zheng

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

About the authors

Alex Reznik (editor)
HPE

Rohit Arora
HPE

Mark Cannon
Virtuosys

Luca Cominardi
UC3M

Walter Featherstone
Viavi Solutions

Rui Frazao
Vasona

Fabio Giust
NEC

Sami Kekki
Huawei

Alice Li
Vodafone

Dario Sabella
Intel

Charles Turyagyenda
InterDigital

Zhou Zheng
Huawei

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 the data closer to the user.

ETSI MEC (Multi-access Edge Computing) Industry Specification Group (ISG) 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 like augmented reality, connected vehicles, etc. and various services can be customized according to the customer requirements and demands.

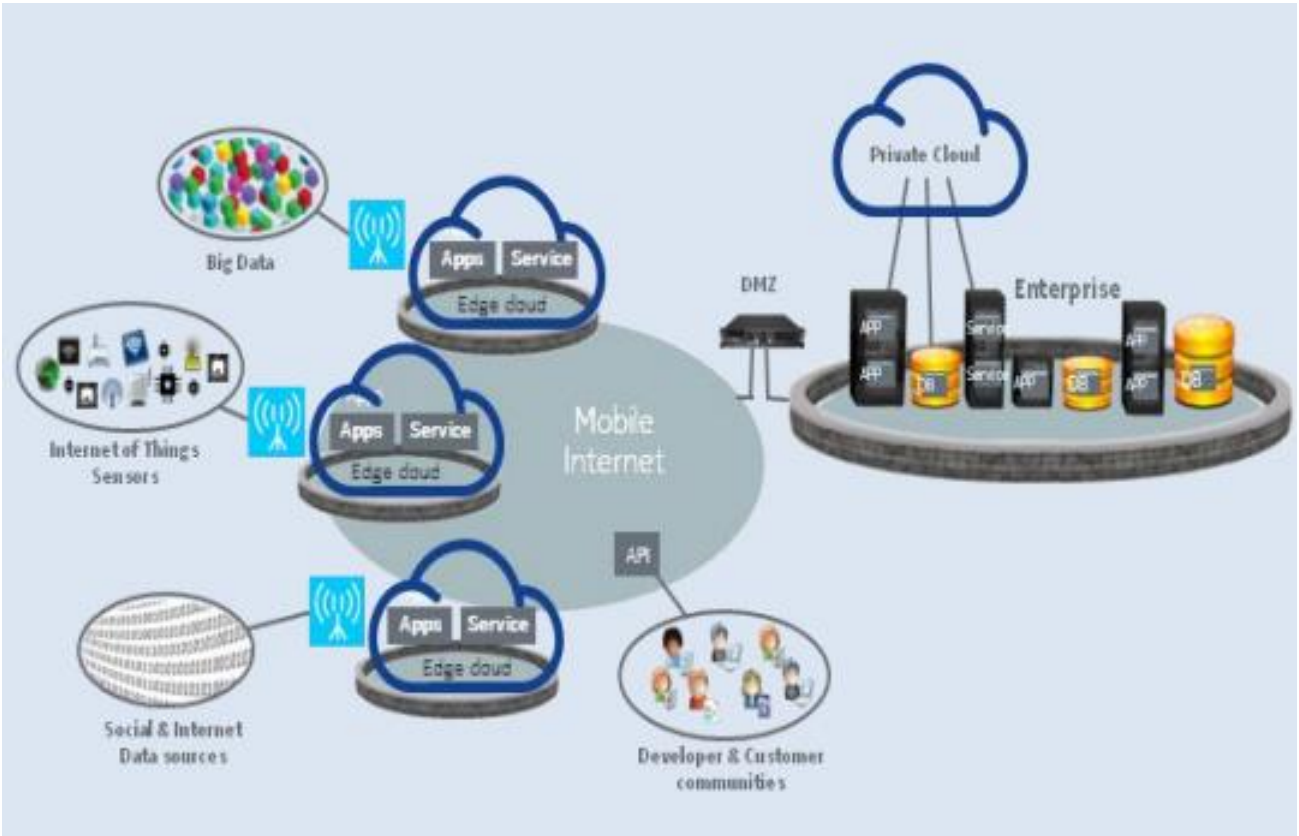

Figure 1: Overview of the MEC system

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.

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 also 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 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. As a consequence, 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 the majority of 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 provides compute, storage and network resources. The MEC platform offers a secure environment where MEC applications may, via RESTful APIs, discover, advertise, consume and offer services.

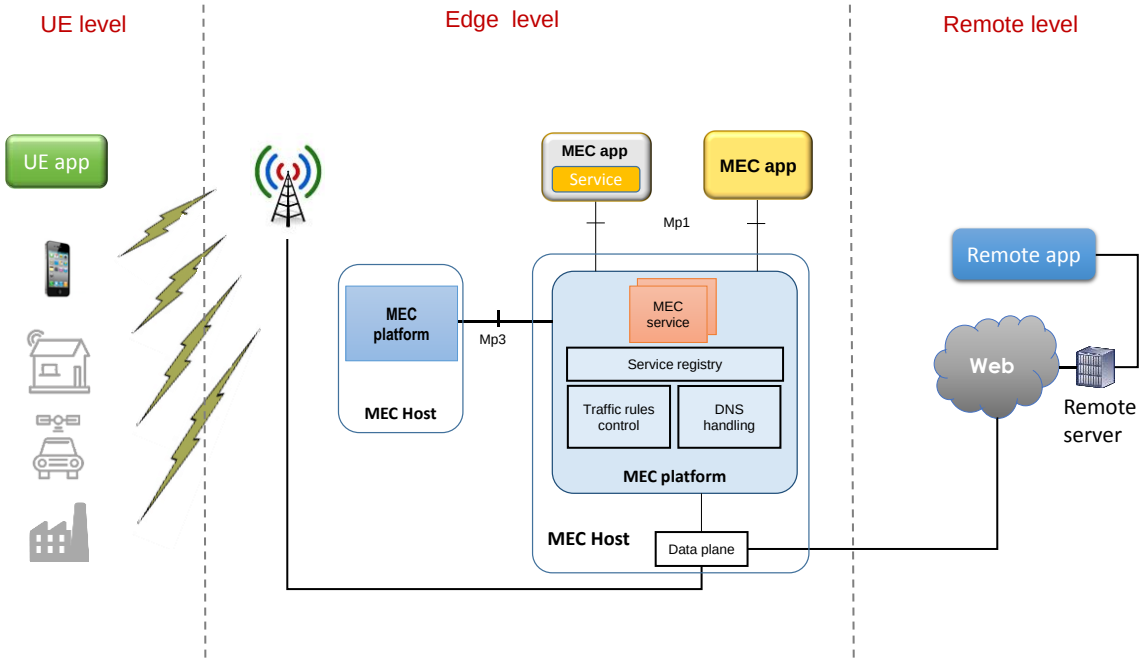

Figure 2: New application development paradigm introduced by MEC.

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 taking into account 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 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) 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. The network becomes a resource used to deliver the end to end service. For example, an application is able to not only precisely monitor the radio link via the MEC Radio Network Information Service, but also make a bandwidth request and inform the network of other application requirements via APIs provided by the MEC platform. This allows edge applications to benefit from low latency and high throughput in a fairly predictable/controllable way. 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.

A key aspect of software design for MEC is the need to split the application into *terminal device component(s)*, *edge component(s)* and *remote component(s)*. This aspect creates an additional task for developers 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. 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. The remote components implement operations to be carried out in the remote data centre, e.g. to benefit from large storage and database access. This concept should not be confused with

the traditional software modularization and distribution of components. This latter vision takes into account the division of tasks, e.g. to improve the development and maintenance of the code, but, usually, the different components either run in the same data centre, or are sparsely distributed at unpredictable locations (e.g. P2P applications). Software modularization is not only possible in MEC, but also encouraged in order 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.

An example of the concept expressed above comes from applications for video analytics from surveillance cameras. Here, the task is to extract information for face recognition from different video streams. In a traditional approach, the video streams are transported up to the data centre where the software components for face recognition, databases, etc. are located. A lot of bandwidth can be saved by splitting the job and performing some functions in the edge or even in the terminal (if the terminal has the necessary computational power). For instance, image areas with suspected faces can be cropped from HD images, compressed, and then sent to the remote cloud, for further processing (i.e. final face identification) and database lookup. When cameras are active, they are constantly searching for pre-defined objects or events. To avoid the need to stream the video constantly in real time to the customer's facilities (e.g. central cloud) the terminal device can do some preliminary processing. Based on the preliminary processing, further action may be necessary on the selected objects, e.g. further analysis of the object's track, detailed identification of the object, etc. Such processing may be delegated to the edge cloud component of the application. The example is illustrated in Figure 3.

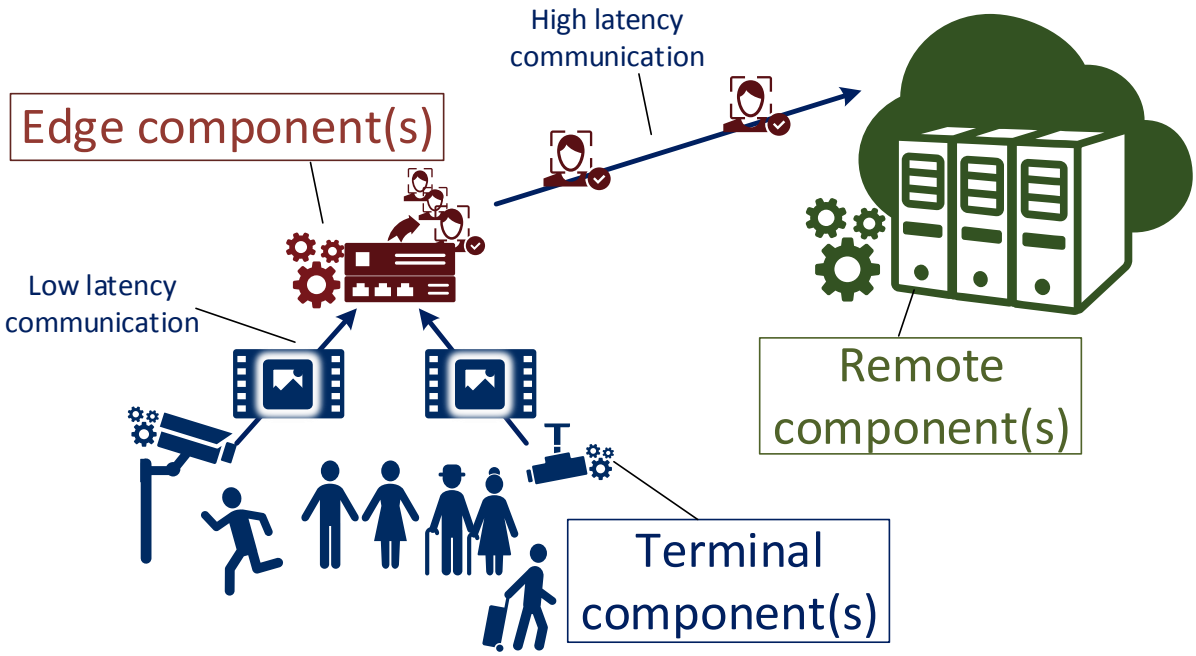

Figure 3: Example of splitting an application into "terminal", "edge" and "remote" components

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.

However, the concept above should not be disguised as yet another load balancing technique. In its traditional sense, load balancing stands for replicating many instances of the application's server side in order to increase the number of clients in service at the same time. In MEC, load balancing can be performed within the same server instance in order to exploit characteristic of the environment where the server component is running.

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 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

ETSI MEC defines a platform that interfaces with MEC applications and other platforms. The MEC platform is responsible for a myriad of functions, including: receiving DNS records and configuring a DNS proxy/server; hosting MEC services, such as: Radio Network Information, Location and Bandwidth Manager. Each application instantiated at the edge would likely only utilize a subset of the totality of functions provided by the MEC platform(s).

Developing a MEC application using monolithic software design principles requires integration of the MEC application and a subset of MEC platform functions into a self-contained, tightly coupled software program. Updates to any MEC platform or program component would necessitate re-writing and re-deploying the entire application.

In contrast, a microservices-based design paradigm would structure a MEC application as a collection of loosely coupled autonomous services working together, i.e. the MEC platform functions could be implemented as microservices with each microservice as a separate entity with no dependency on other microservices forming the MEC application. The microservices interact with each other to implement the logic of the application and communicate via network calls to enforce separation and avoid tight coupling. Consequently, each microservice exposes an application programming interface (API) for other microservices to communicate in collaborative way. One of the most common API paradigms is Representational State Transfer (REST) that provides a uniform and predefined set of stateless operations to access and manipulate resources. The benefits of microservices include.

- A single microservice can be deployed independently of the rest of the system. This allows to deploy some of the components at the edge while keeping others at distinct locations (e.g., at the cloud);
- Microservices permit the use of different technologies inside each microservice and to freely replace the technology stack while the API towards the other microservices remains the same;
- Adapting existing microservices applications does not require refactoring of the whole application;
- Microservices permit scaling of only those microservices that need scaling while keeping the rest of the application untouched;
- Microservices provide better fault isolation, i.e. if one microservice fails, the others continue to function;
- With microservices, it is easy to integrate with 3rd party microservices;
- Microservice-based applications are easily implemented using containers;
- Isolation of the individual microservices simplifies security-related design.

Figure 4, presents the microservices architecture. Each microservice manages its unique data and exposes an API to other microservices. It is worth noting that each microservice may expose a different type of API. Additionally, an API gateway may be used to convey MEC application requests to the appropriate microservice.

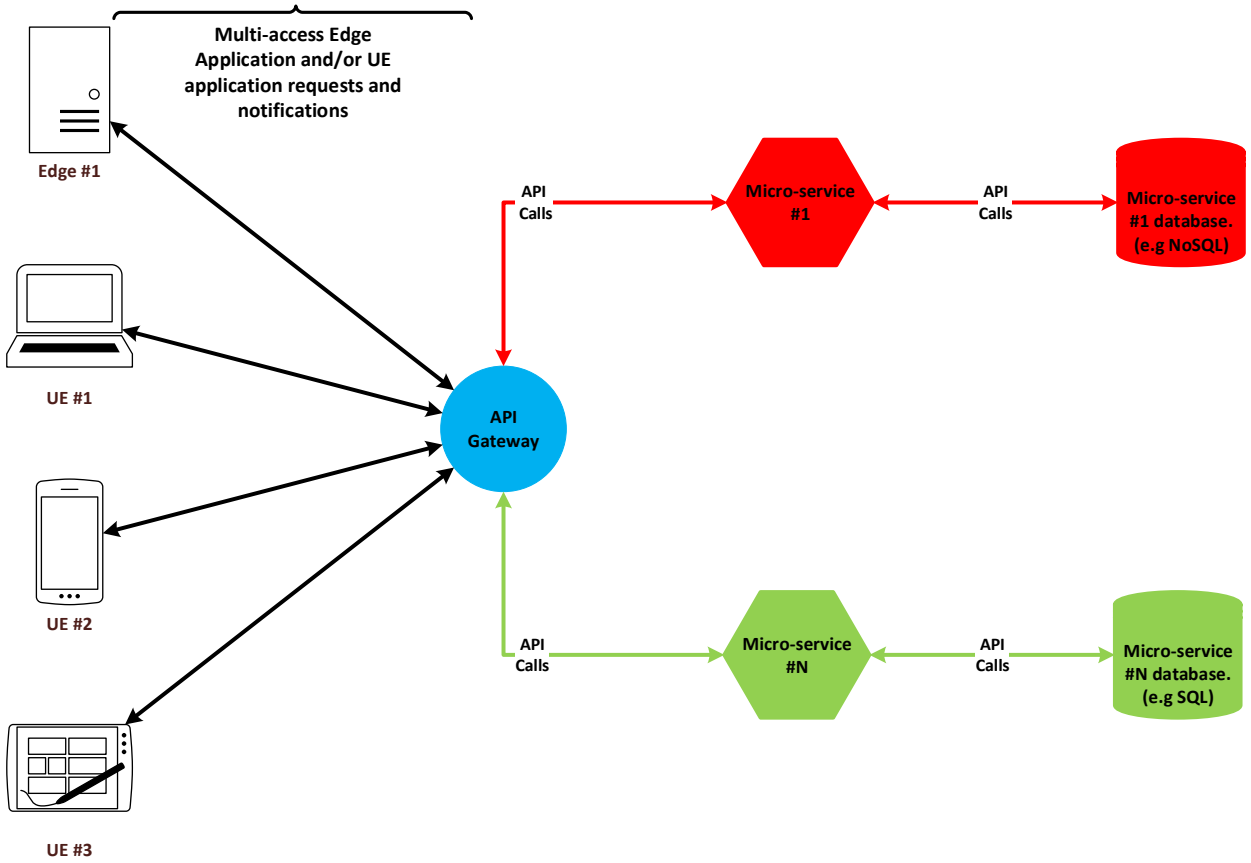

Figure 4: Microservices architecture

Consider an MEC augmented reality application composed of three microservices: one on the client-side, one at the edge, and one at the cloud. The first microservice takes care of collecting the input from the device (e.g., a smartphone) and communicating the data to the microservice residing at the edge. The microservice on the client is also responsible for receiving the computed data from the edge and to displaying it to the user. An API gateway is used in this case to allow the augmented reality logic to support multiple visualization targets at once (e.g., smartphone, tablet, laptop, etc.). The microservice running at the edge is responsible for processing the inputs from the client and combining them to create the augmented reality. This microservice may subscribe to MEC services to improve the quality of experience of the end-user. For example, the microservice may subscribe to the MEC Radio Network Information Service to react to the channel status of the user and enable/disable some augmented reality features depending on the connection quality. Finally, the cloud microservice is responsible for collecting statistics of the users using the augmented reality application.

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.

DevOps is considered to be a business-driven software delivery approach. It 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.

Other Issues

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 similar to other modern applications in terms of security. Approaches, based on modern concepts, such as “zero-trust networking” should be used. In particular, 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 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.
- 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.

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 may not be the best choice for the user due to user mobility. The MEC system allows to relocate the user context from one application instance to another running in a MEC host closer to the user. Furthermore, the MEC system implements a mechanism to relocate the application instance as a whole if necessary. Several use cases for application relocation are documented in ETSI GR MEC 018 “End to End Mobility Aspects”. In these regards, programmers might design their applications with certain capabilities in order to leverage and optimize the UE relocation procedure. Such capabilities might be monitoring the application KPIs to assist the relocation decision, mechanisms to determine the right relocation timing, data synchronization, etc.

Concluding Remarks

This paper addresses 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 provides 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.

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