µMEC in 2019 and plan for 2020

What happened in 2019

In 2019, the µMEC project developed in steps in connection with different events which gave insight into what µMEC should do:

- The **IoThon** in May 2019 was the first event. It was the first time we had the miniature city with Nokia headquarters. We used Ruuvi tag sensors, a webcam, Raspberry Pis and a server. All collected data was stored in an InfluxDB and we gave access to that data. The web camera used a license plate recognition system and stored all recognized license plates to the database. It was not possible to run code on the Raspberries, but the IoThon participants fortunately bypassed that limitation.
- **Junction** and **Akraino ETSI MEC Hackathon** were next to each other which posed new challenges. Also, we had to be ETSI compliant for the second event which meant that we had to be compliant also in the first one. We used remote access to the miniature city in both cases, and this time it was possible to run code on the µMECs. The concept for Junction was to use a user interface to upload containers to a Raspberry Pi k3s cluster, where they would have access to different sensors. All APIs were REST based and there was an ETSI MEC-11 type service registry. It was also possible to get information from the sensors outside of the cluster. This would support applications running in a cloud.
- The time in **Akraino ETSI MEC Hackathon** was much shorter, and therefore there was a ready made sample application. The concept was to write platform independent code (html, JavaScript) that would be run in the cluster by using a web server. There was also a Tensorflow model available for recognizing images. The sample application would be accessed with a browser and would take a picture, which would then be forwarded to a Tensorflow model for classification.

The goal of participating in the Akraino Release 2 was unfortunately not met, although there is now a better understanding of how it should be done. The idea was to use a Metropolia University innovation project to meet the Release 2 deadline which essentially meant building a CI/CD system. The student project would create a test lab that would be connected to the Internet and have a number of Raspberry Pis as test µMECs. There would be an information panel, sample applications, and ETSI compliant APIs to sensors and other services. Many of these goals were met but not all, so no release.

Insights from hackathons

One of the purposes of the Hackathons and other events was to figure out what would be the ideal applications on the platform.

- From the Iothon already, it became clear that **Image processing** is needed. All of the interesting applications used some kind of image processing. In retrospect, image processing is also pretty much the only use case for having compute power at the ultra far edge.
- One of the biggest challenges both for setting up the hackathons and for the participants was **cross compilation**. A Raspberry Pi is so low powered that compiling on it is not really fun. Containers make cross compilation even more difficult: while finding a cross compiler for glibc based distros is easy, the popular Alpine Linux uses musl which requires its own compiler (or compiler settings). If we talk about distroless base images, the build system gets even more complicated. On Raspberry Pis, most distros are 32-bit (SUSE Linux supports 64 bits on Raspberry 3B's and soon on 4B models) while Docker images exist for ARM64. ARMv7 processors may or may not have an FPU, and it sometimes has to be told to the compiler (see https://gist.github.com/TapioT/9ac8e546904447e66a96a91b6f3b1d79).
- We naturally migrated to a **microservice architecture**. It is simply easiest for everyone to write their piece of code in whatever language they want, use whatever base distro they like, and then package the result as a container and push it to Docker Hub. The result can then be installed with docker run.
- As a nice surprise, the **ETSI MEC specification** turned out to be mostly useful. Students in the Metropolia project created an OpenAPI specification for taking pictures and generated the stub code. They were then able to add the real functionality and then they had a working app. The ETSI MEC will likely become more useful once we have more hardware to support.
- **Serverless architectures** seem to be the way to go, supporting the goal of easy and secure development. The development is easy, since OpenFaaS hides the routine parts of development. It is secure since it reduces the attack surface that an application has.

Plan for 2020

Tasks

Here are some of the tasks for 2020, mainly based on the work that has already been done.

- Write the architecture document. We have the pieces already – k3s, Service Registry, developer user interfaces, OpenFaaS serverless, MEC APIs, integration with kubernetes services – but we need to write those up.
- Automatic builds. Since everything in µMEC is built on k3s (kubernetes), we should not worry about OS installation and instead use VMs or containers. One strategy is this:
    - Build all needed containers whenever the code changes
    - Integrate the whole stack: install k3s install OpenFaaS and its dependencies install components run tests
- Once this works, the installation document is easy to write
- Support the specific use cases like Neutral Host
- Trusted computing. We need to make the platform secure
- Integrate OpenFaaS and OpenFaaS Cloud to make application management easy and secure

Vision

The µMEC concept is to have an open server at the ultra far edge that leverages the ETSI MEC architecture. We want to make it secure and easy to develop to. We want to have running code in the miniature city and a working CI/CD system. We want to focus on the concrete use cases such as Neutral Host.