TinyML as-a-Service: Backend Porting Guide

Hiroshi Doyu <hiroshi.doyu@ericsson.com>

Contents

Design

TinyML as-a-Service (TinyMLaaS) would demonstrate ML inference orchestration on IoT devices. TinyMLaaS is just a thin REST API server, based on Swagger UI, which backend would run a requested docker image to build an device OS image on server. Conceptually it's similar to OpenFaaS. Each API call is equivalent to run a certain docker image on server. Those docker images would compile ML inference and then generate an device OS image with the generated ML inference in it. Pre-trained ML inference models are stored in model **Zoo** archive on Cloud. Those OS build docker images are expected to be provided by partners (AI chip / ML compiler vendors). It's done to install those generated device OS images onto devices, via OMA LwM2M FOTA (Firmware Other The Air) update with an URL pointing to an OS image stored on TinyMLaaS. Please refer to Figure 1.

4 main components

This document will explain how TinyMLaaS is to be developed independently by different entities. TinyMLaaS includes 4 components:

1. WebApp
2. API server
3. device OS builder (+ML compiler)
4. device OS downloader (LwM2M FOTA protocol)

For partner to implement backend, only "3. device OS builder (+ML compiler)" is needed.

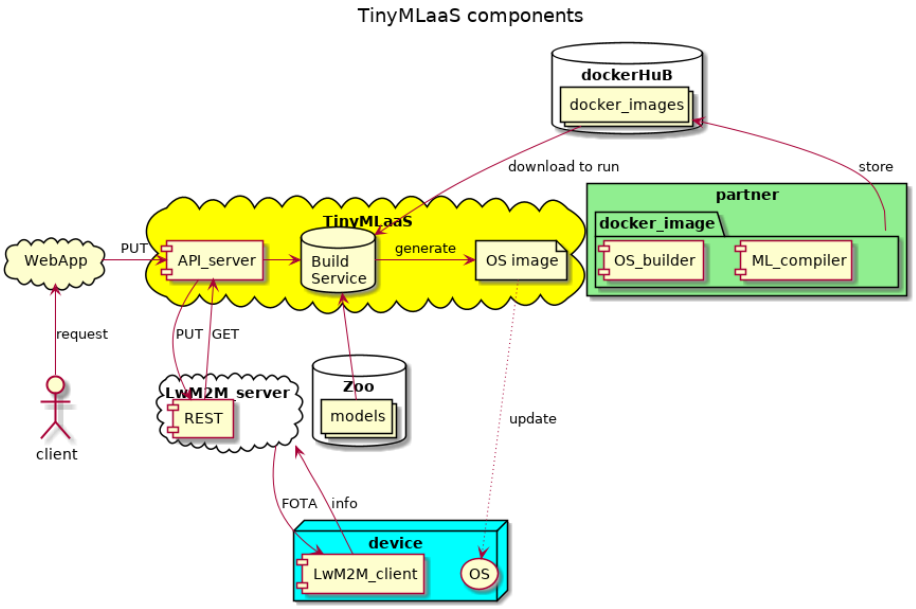

Figure 1: TinyML components

WebApp (TBD)

This is a Frontend GUI for users. Once API server (backend) is implemented, both WebApp (+Mobile App?) could be implemented just by calling those REST API. WebApp could provide nice UI for demonstration, but this could be done later. since all REST API can be called via Swagger UI. Test could be done via Swagger UI without WebApp (Web interface).

API server (Ericsson)

TinyMLaaS is a REST API server. This server side stub is automatically generated by Open API Specification (OAS) file, hosted by Swagger HUB. This also could generate client side code too if needed. The backend of API server would simply run the specified docker image, stored in DockerHUB. This docker image accepts some parameteres passed via REST. For example, target device type and inference model name. Some appropriate docker API is called via Unix domain socket of Docker to run a docker image. Please refer to Figure 2.

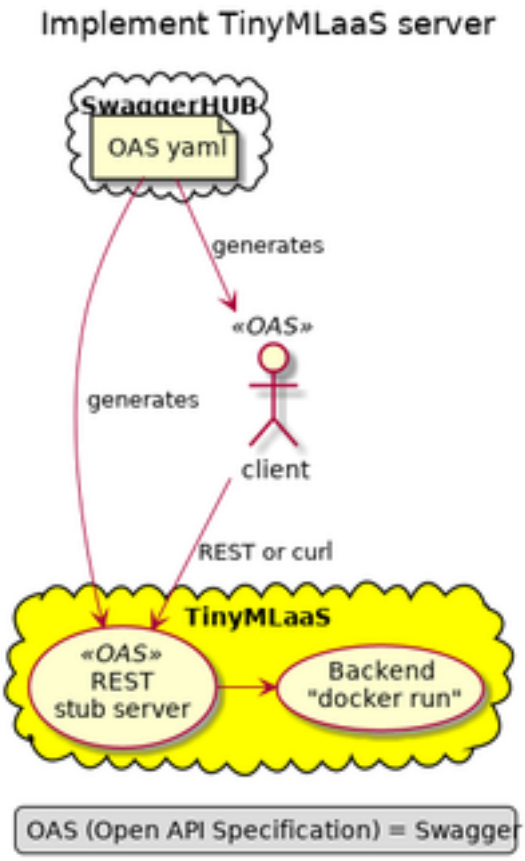

Figure 2: API server

device OS builder (Partner)

This is the part which partner needs to implement. OS build procedure is packed in a docker image. If you pull and run this docker image, it would generate a new OS image, which include a specified ML inference model. Those docker images are provided by partner companies. Those docker images are stored in DockerHUB. We'll support a limitted number of target boards. Device OS should support LwM2M client and its FOTA flashing to re-flash OS image. Some of popular RTOSes support LwM2M with FOTA (FreeRTOS?). LwM2M FOTA (protocol) is the specification of CoAP packet sequence. but it doesn't define how to flash OS image and how to reboot a device. We name this **FOTA flashing**. Those need to be implemented here. A generated OS images are stored in TinyMLaaS. It could be downloaded by http from LwM2M server later. docker image should be run independently and generate OS image with ML inference locally on your host. Please refer to Figure 3.

docker commands to support for unit testing

```
$ docker build -t esp32_wroover .
$ docker run esp32_wroover mnist.model
$ docker push esp32_wroover
```

4 sub-components to implement

To summarize, the following 3 needs to be implemented.

1. ML compiler in docker
2. OS builder in docker
3. LwM2M client in OS
4. OS updater

"1" and "2" can be in a single docker image.

Design option

Implementing FOTA flashing might be too big work. Here are 2 workarounds.

RPI as LwM2M client with flasher to microcontroller

Insert RPI between LwM2M server and microcontroller device. Let RPI pretend to include microcontroller device. It's easy to support LwM2M client (protocol) && FOTA flashing on RPI. For demonstration, RPI doesn't look nice, but still

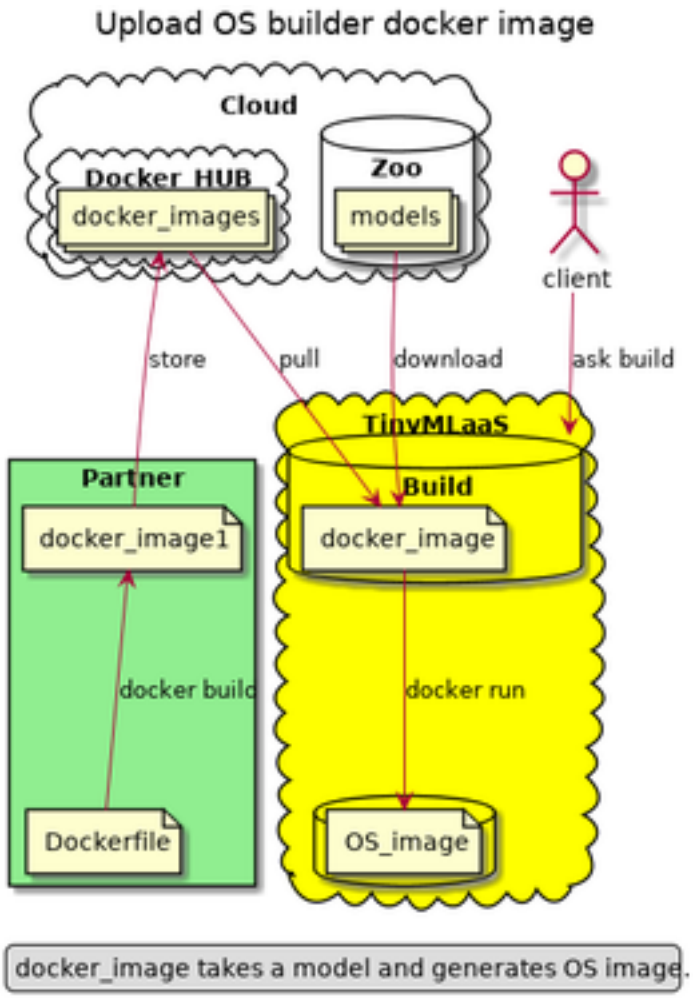

Figure 3: device OS builder

this could be used as transitional development environment until device support FOTA flushing. Please refer to Figure 4.

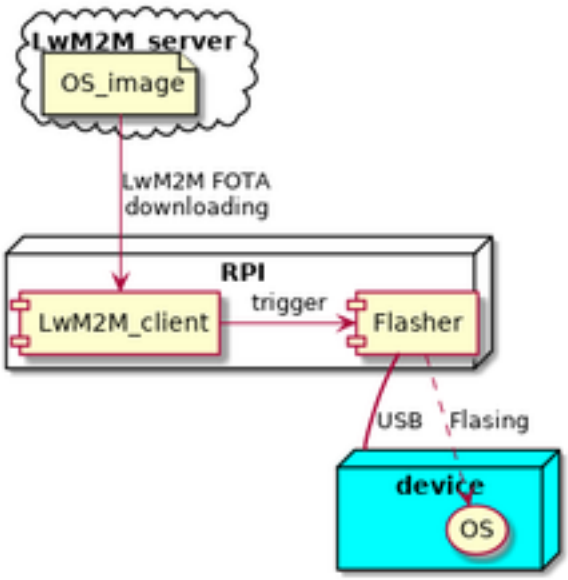

Figure 4: RPI as LwM2M client with flasher to microcontroller

LwM2M SOTA (Software Over The Air) update

Inference module could be stored in certain ELF section. LwM2M client SOTA would update only this section. SOTA doesn't require full flash but only updating a section partially. SOTA doesn't stop a working system. This may be easier if LwM2M is already supported by the OS you use.

device OS downloader (LwM2M FOTA)

LwM2M FOTA (protocol) would replace a whole OS image with updated ML inference model. Leshan server(LwM2M server) has published a REST API for FOTA protocol. A client would trigger FOTA protocol via the above Leshan REST API. How to flash OS image and how to reboot a device are implemented in the previous device OS builder phase. There's no implementation work here. Please refer to Figure 5.

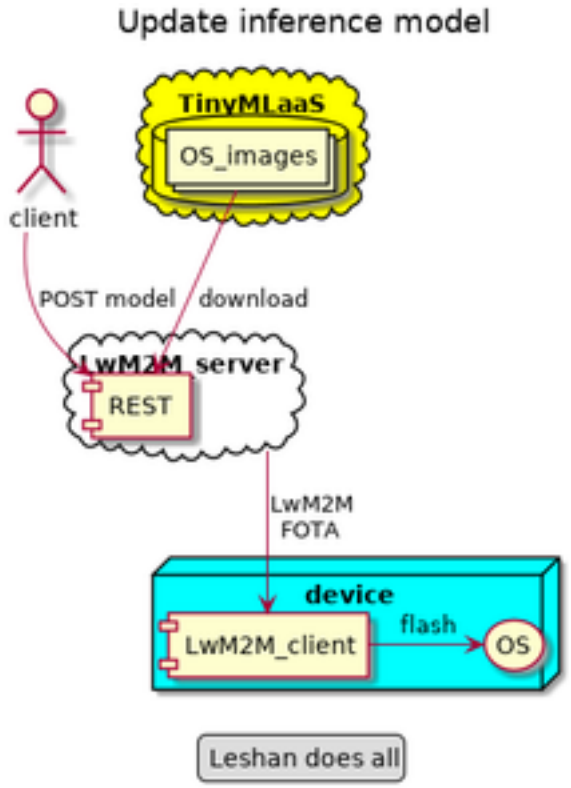

Figure 5: device OS updater

Use case

This demo is mainly about ML inference orchestration. Use case depends highly on the feature of target boards partner uses. Please provide the following info:

1. Which target boards to use?
2. Which ML inferences to replace?
3. What kind of use case scenario?

If you have any questions, don't hesitate to ask any questions.