

소프트웨어 업데이트

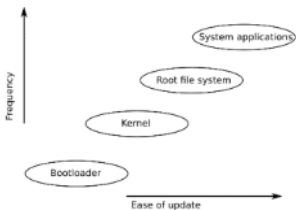
소프트웨어 업데이트 방법

- 1. 로컬 업데이트:
USB 플래시 드라이브나 SD 카드 같은 휴대용 매체에 담아 각 시스템에 기술자가 개별적으로 접근해서 수행
- 2. 원격 업데이트:
사용자나 기술자가 현장에서 수행하지만, 다운로드를 원격 서버로부터 받음
- 3. OTA 업데이트:
현장 작업 없이, 업데이트가 완전히 원격에서 전달되고 관리됨

무엇을 업데이트해야 하는가?

임베디드 리눅스 장치의 기본 요소들

- 부트로더
- 커널
- 루트 파일시스템
- 시스템 애플리케이션
- 장치별 데이터



<부트로더>

프로세스 전원을 켤 때 실행되는 첫 번째 코드

업데이트 도중 문제가 발생하면 위험하기에 백업이 없다면 대부분의 경우 업데이트하지 않고 다른 부분만 업데이트하는 것이 좋음

<커널>

리눅스 커널은 수시로 업데이트해야 하는 핵심 요소로 여러 부분으로 나뉘어 있음

- 부트로더가 로드한 바이너리 이미지는 보통 루트 파일시스템에 저장
- 대부분의 장치에는 하드웨어에 대한 정보를 커널에 알려주는 장치 트리 바이너리(DTB, Device Tree Binary)가 존재, 함께 업데이트 돼야 함 (DTB는 보통 커널 바이너리와 함께 저장)
- 루트 파일시스템에 커널 모듈이 있는 경우도 있음

커널과 DTB는 부트로더가 해당 파일시스템 형식을 읽을 수 있는 기능을 갖고 있는 경우 루트 파일시스템에 저장돼 있을 수 있으며, 그렇지 않은 경우 지정 파티션에 저장됨
두 경우 모두 백업 사본을 갖는 것이 더 안전

<루트 파일시스템>

루트 파일시스템에는 시스템을 작동하는 데 필요한 필수 시스템 라이브러리, 유틸리티, 스크립트가 들어있음
이 모든 파일은 교체하고 업그레이드 하는 편이 좋음

임베디드 루트 파일시스템의 일반적인 형식은 다음과 같음

램디스크:

원시 플래시 메모리나 부팅 시 디스크 이미지에서 로드됨
업데이트 하려면 램디스크 이미지를 덮어쓴 후 재부팅

플래시 파티션에 저장된 squashfs 같은 읽기 전용 압축 파일시스템:

이러한 유형의 파일시스템은 쓰기 기능을 지원하지 않으므로 이를 업데이트 할 수 있는 유일한 방법은 전체 파일시스템 이미지를 해당 파티션에 업데이트 하는 것

일반 파일시스템 유형:

원시 플래시 메모리의 경우 JFFS2나 UBIFS 포맷이 일반적이며 eMMC나 SD 카드처럼 컨트롤러가 내장된 플래시 메모리의 경우 ext4나 F2FS 포맷일 가능성이 큼
이들은 런타임에 쓰기가 가능하므로 파일별로 업데이트할 수 있음

<시스템 애플리케이션>

시스템 애플리케이션은 장치의 주요 구성 요소로 장치의 기본적인 기능을 구현 버그를 수정하고 기능을 추가하기 위해 자주 업데이트 될 수 있음 이들은 루트 파일시스템에 함께 저장될 수 있지만, 업데이트를 쉽게 하고 오픈소스 시스템 파일과 저작권이 있는 애플리케이션 파일을 분리하기 위해 별도의 파일시스템에 저장되는 경우가 더 일반적

<장치별 데이터>

런타임에 수정되는 파일 모음으로 구성 설정, 로그, 사용자 제공 데이터 등이 포함 자주 업데이트를 해야될 필요는 없지만 업데이트 중에 보존돼야만 하는 데이터로 자체 파티션에 저장

소프트웨어 업데이트의 기본

소프트웨어 업데이트 매커니즘은 최소한 다음과 같은 조건을 따라야 함

- 견고 해야함
- 안정장치가 필요함
- 보안이 필요함

<견고한 업데이트>

업데이트는 전체적으로 원자성을 지켜야 함 (즉, 시스템의 일부를 업데이트할 때 다른 부분을 건드리지 말 것)

원자성을 달성하려면, 실행 중인 시스템과 함께 업데이트를 설치한 후 기존 시스템에서 새 시스템으로 변경 돼야 함 원자성을 달성하기 위한 두가지 접근 방식

- 루트 파일시스템과 다른 주요 컴포넌트의 복사본을 2개 갖는 것 (하나는 현재 실행 중인 시스템, 다른 하나는 업데이트를 받을 수 있는 시스템) 이 경우 업데이트가 완료되면 스위치가 실행돼 재부팅 시 부트로더가 업데이트된 복사본을 선택하는 것 이를 시메트릭 이미지 업데이트나 A/B 이미지 업데이트라고 함 원자성은 부트로더와 복구 운영 체제 간에 공유돼 보장 이를 어시메트릭 이미지 업데이트라고 하며, 누가 7.x 버전 이전에 안드로이드에서 사용한 방식
- 시스템 파티션의 각기 다른 서브 디렉터리에 루트 파일시스템을 2개 이상 복사한 다음 부팅시 chroot (8)을 사용해 그중 하나를 선택하는 것 리눅스가 실행되면 업데이트 클라이언트가 다른 루트 파일시스템에 업데이트를 설치하게 되고, 모든 것이 완료되고 확인되면 기존 시스템과 신규 시스템을 바꾸는 스위치가 실행돼 재부팅 이를 원자 파일 업데이트라고 함

<안전한 업데이트>

제대로 설치됐지만 해당 업데이트로 인해 시스템이 제대로 부팅되지 않는 경우 가장 이상적인 방법은 시스템이 이러한 문제를 감지하고 제대로 동작한 이전 이미지로 되돌리는 것

시스템이 동작할 수 없는 상태로 이어질 수 있는 몇가지 장애 모드 존재

- 커널 장치 드라이버의 버그나 init 프로그램을 실행할 수 없는 경우 발생하는 커널 패닉 <해결 방법>
 - 패닉이 일어나면 이후 몇 초 만에 재부팅 할 수 있도록 커널을 구성하는 것 (CONFIG_PANIC_TIMEOUT을 설정해 커널을 빌드하거나 커널 명령 줄에서 panic 값 설정하기)
 - 옵스에서 패닉이 일어나도록 커널을 설정하기 (커널 설정에서 CONFIG_PANIC_ON_OOPS = y로 설정하거나 커널 명령줄에서 oops = panic으로 설정)
- 커널이 init을 성공적으로 시작하기는 했으나 어떤 이유로든 주 애플리케이션이 실행되지 않을 때 발생 <해결 방법>
 - 워치독 (watchdog) 필요 워치독이란 타이머가 만료되기 전에 리셋되지 않으면 시스템을 다시 시작하는 하드웨어나 소프트웨어 타이머

위 두가지 장애 모드 모두 시스템 부팅이 반복 (커널 패닉과 워치독 타임아웃 모두 시스템을 재부팅 시킴) 무한 부팅 루프에서 벗어나려면, 부트로더가 이러한 장애를 탐지해 정상적으로 동작하던 이전 버전으로 시스템을 되돌리는 코드 필요 가장 일반적인 방법은 부트로더가 매 부팅 시 추가하는 부팅 횟수를 이용하는 것 시스템이 부팅을 반복하게 되면 횟수는 증가됨 특정 임계 값을 초과하게 되면 부트로더가 적절한 복구 작업을 수행하도록 구성하면 됨

U-Boot에서는 세가지 변수로 이를 처리

bootcount	프로세스가 부팅할 때마다 증가
bootlimit	Bootcount가 bootlimit을 초과하면 U-Boot는 bootcmd 대신 albootcmd에서 명령어 실행
altbootcmd	기존 부팅 명령어를 대체하는 명령어 포함

이 방법을 사용하려면 사용자 공간 프로그램이 부팅 횟수를 리셋할 수 있어야 함 다음 U-Boot 유틸리티를 사용하면 런타임 시 U-Boot 환경에 접근할 수 있음

fw_printenv	U-Boot 변수 값을 출력
fw_setenv	U-Boot 변수 값을 지정

위 두 명령어를 사용하려면 U-Boot 환경이 블록에 저장돼 있는지 알아야 하기에, /etc/fw_env.config의 설정 파일 확인

부팅할 때 마다 부팅 횟수를 하나씩 늘린 다음에 애플리케이션이 실행될 때 이를 재설정하게 되면, 환경 블록에 이 값을 불필요하게 여러 번 작성하게 됨 (메모리 소모, 속도 저하 초래)

U-Boot가 재부팅할 때마다 이러한 작업을 하는 것을 방지하기 위해 upgrade_available이라는 변수가 있음
만약 upgrade_available이 0이면 bootcount 값이 증가하지 않음
Update_available은 업데이트가 설치된 후 1로 설정되므로 필요할 때만 bootcount 가 보호됨

<업데이트 보안>

보안 패치와 새로운 기능을 설치하는 업데이트 매커니즘을 구현할 때 가장 중요한 것은 신뢰할 수 있는 자동화나 반자동화된 방식을 제공하는 것

업데이트 보안의 가장 큰 취약점은 가짜 원격 업데이트
이를 방지하려면, 다운로드를 시작하기 전에 업데이트 서버를 인증해야 하며 다운로드 스트림을 변조하지 못하도록 HTTPS 같은 보안 전송 채널을 사용해야 함

로컬로 제공되는 업데이트 또한 신뢰성에 대한 확인이 필요
가짜 업데이트를 찾아내는 방법 중 하나는 부트로더에 있는 Secure boot 프로토콜을 사용하는 것
U-Boot 소스 코드 확인법 (doc/ulmage.FIT/verified-boot.txt)

업데이트 매커니즘 유형

소프트웨어 업데이트를 적용하는 세가지 방법

1. 시메트릭(혹은 A/B) 이미지 업데이트
2. 어시메트릭 이미지 업데이트 (복구 모드 업데이트)
3. 원자 파일 업데이트

<시메트릭 이미지 업데이트>

이 방법은 운영체제의 사본을 2개 갖고 있어야 하며 각각 리눅스 커널, 루트 파일시스템, 시스템 애플리케이션으로 구성

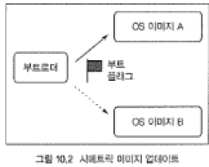


그림 10.2 시메트릭 이미지 업데이트

시메트릭 이미지 업데이트의 동작

1. 부트로더는 어떤 운영체제를 로드해야 할지 알려주는 플래그를 가지고 있음
처음에는 플래그가 A로 설정돼 부트로더가 OS 이미지 A를 로드
2. 업데이트를 설치하려면, 운영체제의 일부인 업데이트 애플리케이션이 OS 이미지 B를 새롭게 덮어씀
3. 해당 작업이 완료되면 부트 플래그가 B로 변경되고 재부팅
4. 부트로더가 새로운 운영체제에 로드
5. 추가적인 업데이트를 설치한다면, 업데이트는 이제 이미지 A를 덮어 쓰고 부팅 플래그를 A로 변경
두 운영체제 복사본 간에 평풍이 계속 일어나게 됨
6. 만약 부트 플래그가 변경되기 전에 업데이트가 실패하면, 부트로더는 계속해서 기존의 양호한 운영체제를 로드

<단점>

1. 전체 파일시스템 이미지를 업데이트하면 업데이트 패키지의 크기가 커서 장치를 연결하는 네트워크 인프라에 부담을 줄 수 있다는 것
(이전 버전과 새 파일시스템의 바이너리 diff를 수행해 변경된 파일시스템 블록만 전송하면 문제 완화 - 델타 업데이트)
2. 루트 파일시스템과 다른 구성 요소의 사본을 중복해서 저장해야 하므로 쓸데없이 많은 저장 공간을 유지할 필요가 있다는 것
(루트 파일시스템이 가장 큰 구성 요소인 경우 사용해야 할 플래시 메모리의 양이 두배가 될 것임, 이때문에 어시메트릭 업데이트 체계 사용)

<어시메트릭 이미지 업데이트>

최소한의 복구 운영체제만으로 주 운영체제를 업데이트하면 필요한 저장 공간을 줄임

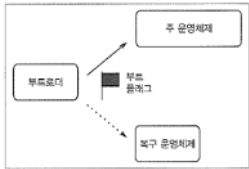


그림 10.3 아시메트릭 이미지 업데이트

아시메트릭 업데이트 방법

- 부트 플레그가 복구 운영체제를 가리키도록 설정하고 재부팅
- 복구 운영체제가 실행되면 주 운영체제 이미지로 이 업데이트를 스트리밍할 수 있음
- 만약 중간에 업데이트가 중단되면, 부트로더가 다시 복구 운영체제로 부팅돼 업데이트를 다시 시작할 수 있음
- 업데이트가 완료되고 확인됐을 때만 복구 OS가 부트 플레그를 지우고 다시 부팅 (새로운 주 운영체제를 로드하게 됨)
- 정상적인 파일이지만 버그가 있는 업데이트인 경우 폴백은 시스템을 복구 모드로 되돌려 놓으며, 이 모드에서는 이전 업데이트 버전을 요청해 복구 작업을 시도

<단점>

복구 OS가 실행되는동안 장치가 동작하지 않는다는 것
 이러한 업데이트 체계는 복구 OS 자체의 업데이트를 허용하지 않음
 이 경우는 A/B 이미지 업데이트와 같은 것이 필요하므로 이 업데이트 체계를 사용하는 목적 자체를 무효화

<원자 파일 업데이트>

부팅 시 chroot (8) 명령을 사용해 한 파일시스템의 여러 디렉터리에 존재하는 루트 파일시스템의 여러 복사본 중 하나를 선택하는 것
 다른 디렉터리가 root 디렉터리로 마운트되는 동안 다른 디렉터리 트리를 갱신할 수 있음
 루트 파일 시스템의 버전이 바뀔 때 변경되지 않는 파일의 경우 해당 파일의 사본을 만드는 대신 링크를 사용할 수 있음
 (디스크 공간 절약, 데이터양 줄일 수 있음)

OSTree 프로젝트 (libOSTree)

타킷의 /ostree/repo/objects 디렉터리에 파일 저장
 동일한 파일의 여러 버전이 저장소에 존재할 수 있도록 파일명이 지정되고, 해당 파일 모음이 /ostree/deploy/os/29ff9.../와 같은 이름을 가진 배포 디렉터리에 링크
 각각의 배포 디렉터리는 루트 파일시스템을 구성하는 파일을 포함하고 있으며 기본적으로는 2개 존재

OSTree 디렉터리에서 부팅하는 방법

- 부트로더는 initramfs로 커널을 부팅하고 사용할 배포 디렉터리 경로를 커널 명령 줄을 전달
- Initramfs는 init 프로그램인 ostree-init 을 포함하고 있으며 ostree-init 은 명령줄을 읽고 주어진 경로로 chroot를 실행
- 시스템 업데이트가 설치되면 OSTree 설치 에이전트가 변경된 파일을 repo 디렉터리로 다운로드
- 다운로드가 완료되면, 새 루트 파일시스템을 구성할 파일 모음에 대한 링크가 포함된 새로운 deploy 디렉터리가 생성
- OSTree 설치 에이전트가 부트로더의 부트 플레그를 qsrudg 다음 재부팅 시에 새로운 deploy 디렉터리로 chroot 할 수 있도록 함
- 부트로더는 부팅 횟수를 확인한 후 부트 루프가 감지되면 이전 루트로 폴백

OTA 업데이트

OTA(Over-the-Air) 업데이트는 사용자가 장치를 직접 조작하지 않고 네트워크를 통해 소프트웨어를 자동으로 설치하는 방법

클라이언트가 업데이트 서버를 수시로 폴링(Polling)해 보류 중인 업데이트가 있는지 확인 (이때 폴링 메시지를 서버로 보냄)

배포 및 관리: 서버는 여러 장치에 업데이트를 일괄 배포할 수 있으며, 각 장치의 상태를 모니터링하고 문제가 있을 경우 알림

보안: 업데이트가 악의적으로 변경되는 것을 방지하기 위해, 서버와 클라이언트는 인증서를 교환하고 다운로드한 소프트웨어 패키지가 올바른지 서명을 검증해야 함

OTA 업데이트에 사용가능한 세 가지 오픈소스 프로젝트

- Mender의 Managed 모드
- balena
- SWUpdate나 RAUC 같은 업데이트 클라이언트와 결합된 이클립스 hawBit

로컬 업데이트 시 Mender 사용하기

Mender는 시메트릭 A/B 이미지 업데이트 매커니즘을 사용하며, 실패한 업데이트의 경우 폴백을 지원

로컬 업데이트 => 스탠드 얼론 모드

OTA 업데이트 => Managed 모드

- Mender 클라이언트 빌드하기

폴링 메시지

클라이언트 장치가 업데이트 서버에 주기적으로 보내는 요청 메시지
 업데이트 필요 여부나 장치 상태, 소프트웨어 버전 확인 등

폴링 메시지에 포함되는 주요 정보:

- 장치 고유 식별자 (MAC 주소나 일련번호)
- 현재 소프트웨어 버전
- 장치 상태 정보 (장치의 가동 시간이나 환경 변수 같은 운영 상태 정보)
- 기타 상태 정보 (중앙에서 장치를 관리하는 데 유용한 추가적인 장치 상태 정보들)

스탠드 얼론 모드

Mender 클라이언트는 Yocto 메타 레이어로 사용할 수 있음

- meta-mender 레이어 가져오기 (poky 디렉터리 위의 디렉터리로 이동)

```
ysj@build-master:~/s32g-bsp-300-master$ git clone -b dunfell https://github.com/mendersoftware/meta-mender.git
Cloning into 'meta-mender'...
remote: Enumerating objects: 30319, done.
remote: Counting objects: 100% (4110/4110), done.
remote: Compressing objects: 100% (833/833), done.
remote: Total 30319 (delta 3413), reused 3739 (delta 3246), pack-reused 26209 (from 1)
Receiving objects: 100% (30319/30319), 7.60 MiB | 7.27 MiB/s, done.
Resolving deltas: 100% (19728/19728), done.
ysj@build-master:~/s32g-bsp-300-master$
```

- 빌드 디렉터리를 만들고 레이어 추가

```
$ source poky/oe-init-build-env build-mender-qemu
$ bitbake-layers add-layer ../meta-openembedded/meta-oe
$ bitbake-layers add-layer ../meta-mender/meta-mender-core
$ bitbake-layers add-layer ../meta-mender/meta-mender-deno
$ bitbake-layers add-layer ../meta-mender/meta-mender-qemu
```

- Conf/local.conf 에 다음을 추가해 환경 설정

```
MENDER_ARTIFACT_NAME = "release-1"
INHERIT += "mender-full"
MACHINE = "vexpress-qemu"
INIT_MANAGER = "systemd"
IMAGE_FSTYPES = "ext4"
```

- 이미지 빌드 (빌드 결과는 tmp/deploy/images/vexpres-qemu)

```
$ bitbake core-image-full-cmdline
```

Mender 레이어에서 제공하는 스크립트를 사용하면 QEMU 타겟을 실행할 수 있음

- 이 스크립트는 먼저 U-Boot를 부팅한 다음 리눅스 커널을 로드

```
$ ../meta-mender/meta-mender-qemu/scripts/mender-qemu
[+]
[ OK ] Started Mender OTA update service.
[ OK ] Started Mender Connect service.
[ OK ] Started NFS status monitor for NFSv2/3 locking..
[ OK ] Started Respond to IPv6 Node Information Queries.
[ OK ] Started Network Router Discovery Daemon.
[ OK ] Reached target Multi-User System.
      Starting Update UTMP about System Runlevel Changes...

Poky (Yocto Project Reference Distro) 3.1.6 vexpress-qemu
ttyAMA0

vexpress-qemu login:
```

(로그인 프롬프트 대신 에러 시 시스템에 qemu-system-arm을 설치하고 스크립트 다시 실행)

```
$ sudo apt install qemu-system-arm
```

- 패스워드 없이 root로 로그인

```
# fdisk -l /dev/mmcblk0
Disk /dev/mmcblk0: 608 MiB, 637534208 bytes, 1245184 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 15F2C2E8-D576-4A14-A5F4-4D571185E8D0

Device      Start      End Sectors  Size Type
/dev/mmcblk0p1 16384    49151    32768    16K EFI System
/dev/mmcblk0p2 49152   507903   458752   224K Linux filesystem
/dev/mmcblk0p3 507904   966655   458752   224K Linux filesystem
/dev/mmcblk0p4 966656  1245150   278495   136K Linux filesystem
```

총 4개의 파티션 존재

파티션 1: U-Boot의 부트 파일을 갖고 있음

파티션 2, 3: A/B 루트 파일시스템을 갖고 있음

이 단계에서 두 파티션의 값은 동일

파티션 4: 나머지 파티션을 포함하는 확장 파티션

업데이트 설치하기

루트 파일시스템을 변경해 업데이트 설치

1. 다른 셸을 열고 작업 빌드 디렉터리로 다시 이동

```
$ source poky/oe-init-build-env build-mender-qemu
```

2. 방금 빌드한 이미지의 사본 가져오기 (업데이트 할 라이브 이미지)

```
$ cd tmp/deploy/images/vexpress-qemu
$ cp core-image-full-cmdline-vexpress-qemu-grub.uefimg \
  core-image-live-vexpress-qemu-grub.uefimg
$ cd -
```

3. 타깃의 호스트 이름을 변경해두면 해당 타깃이 설치될 때 쉽게 알 수 있음
Conf/local.conf 에 다음 행 추가

```
hostname_pn-base-files = "vexpress-qemu-release2"
```

4. 이미지 빌드

```
$ bitbake core-image-full-cmdline
```

Core-image-full-cmdline-vexpress-qemu-grub.mender에 있는 새로운 루트 파일시스템만 이용

.mender 파일은 Mender 클라이언트가 인식할 수 있는 포맷이며 버전 정보, 헤더, 루트 파일시스템 이미지를 압축된 .tar 아카이브 구성

5. 새 버전을 타깃에 배포

ctrl+A를 입력한 다음 x를 입력함으로써 이전 터미널 세션에서 시작한 에뮬레이터를 중지해 종료

그 다음, 새롭게 복사한 이미지를 사용해 QEMU 를 재부팅

```
$ ../meta-mender/meta-mender-qemu/scripts/mender-qemu \
core-image-live
```

6. QEMU가 10.0.2.15이고 호스트가 10.0.2.2 인 네트워크가 구성돼 있는지 확인

```
# ping 10.0.2.2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=255 time=0.286 ms
^C
--- 10.0.2.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time
0ms
rtt min/avg/max/mdev = 0.286/0.286/0.286/0.000 ms
```

7. 다른 터미널 세션을 열고, 업데이트를 제공할 수 있는 호스트의 웹 서버를 시작

```
$ cd tmp/overlay/images/vexpress-qemu
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/)
```

...
(ctrl + c를 입력해 종료)

8. 타깃으로 돌아와, 업데이트 받기

```
# mender --log-level info install \
> http://10.0.2.2:8080/core-image-full-cmdline-vexpressqemu-
grub-mender
INFO[0751] Wrote 234883824/234883824 bytes to the
inactive partition
INFO[0751] Enabling partition with new image installed to
be a boot candidate: 3
```

9. QEMU 명령줄에서 reboot를 입력해 QEMU를 재부팅

이제 루트 파일시스템은 파티션 3에서 마운트되며, 호스트 이름은 다음과 같이 변경

fw_printenv를 사용해 U-Boot 변수를 살펴보면 bootcount를 리셋하지 않은 채 시스템이 부팅되면 U-Boot가 이를 감지해 이전 버전으로 롤백 되는 것을 확인할 수 있음

```
# mount
/dev/mmcblk0p3 on / type ext4 (rw,relatime)
[...]
```

```
# hostname
vexpress-qemu-release2
```

U-Boot 플백 동작 테스트

1. 타깃을 바로 재부팅
2. 타깃이 재부팅되면 U-Boot 가 이전 설치 버전으로 되돌린 것을 확인
3. 업데이트 절차를 다시 반복
4. 이번에는 재부팅 후 변경 사항을 commit
5. Upgrade_available이 지워지면 U-Boot는 더 이상 bootcount 값을 확인하지 않으므로 장치는 이 업데이트된 루트 파일 시스템을 계속 마운트
추가 업데이트가 로드되면, Mender 클라이언트는 bootcount를 지우고 upgrade_available 값을 다시 한번 설정

OTA 업데이트 시 Mender 사용하기

Managed 모드

- 도커와 도커 컴포즈가 설치돼 있는지 확인

```
$ docker --version
Docker version 19.03.8, build afac68b7f0
$ docker-compose --version
docker-compose version 1.25.0, build unknown
```

- Mender 서버에 json 파서 설치

```
$ sudo apt install jq
```

- Mender 통합 환경 설치

```
$ curl -L \
https://github.com/mendersoftware/integration/
archive/2.5.1.tar.gz | tar xz
$ cd integration-2.5.1
$ ./demo up
```

```
Starting the Mender deno environment...
[~]
Creating a new user...
*****

Username: mender-demo@example.com
login password: D53444451286

*****
Please keep the password available, it will not be cached by
the login script.
Mender deno server ready and running in the background. Copy
credentials above and log in at https://localhost
Press Enter to show the logs.
Press Ctrl-C to stop the backend and quit.
```

로컬 서버에서 업데이트를 롤링하도록 타깃의 설정을 변경하는 방법

Hosts 파일에 한 행을 추가해 서버 URL docker.mender.io 와 s3.docker.mender.io 를 localhost 주소에 매핑 (Yocto 프로젝트에서 변경하는 방법)

1. Yocto를 복제한 디렉터리의 한 레벨 위 디렉터리로 이동
2. Hosts 파일을 생성하는 레시피에 추가할 파일과 함께 레이어를 생성 (recipe-core/base-files/base-files_3.0.14.bbappend)
3. 작업 빌드 디렉터리 source 하기
4. meta-ota 레이어를 추가
5. 새 이미지 빌드
6. 라이브 이미지로 사용하게 될 복사본 만들기
7. 라이브 이미지 부팅
8. /etc/mender/mender.conf 파일을 보면 폴링 간격이 어떻게 구성돼 있는지 알 수 있음
9. 웹 사용자 인터페이스로 돌아와서 녹색 아이콘을 클릭해 새 장치 인증
10. 장치 항목을 클릭해 세부 정보 확인

OTA 업데이트

1. Conf/local.conf 업데이트
2. 이미지 재 빌드
3. Release 탭을 열고 왼쪽 하단 모서리에 있는 보라색 Upload 버튼을 클릭해 이를 웹 인터페이스로 가져옴
4. Tmp/deploy/images/vexpress-qemu에서 core-image-cmdline-vexpress-qemu-grub.mender 파일을 찾아 업로드

QEMU 장치에서 업데이트 배포

1. Devices 탭을 클릭하고 장치 선택
 2. 장치 정보의 오른쪽 하단에 있는 create a deployment for this device 옵션 클릭
 3. Release 페이지에 있는 OTA-update1 아티팩트를 선택하고 create deployment with this release 버튼 클릭
 4. Create a deployment 의 select target software and devices 단계에서 next 버튼 클릭
 5. 배포 상태는 pending 상태에서 in progress 상태로 바뀜
 6. 약 13분이 지나면 mender 클라이언트는 예비 파일시스템 이미지에 업데이트를 작성하고 QEMU 는 장치를 재부팅한 후 이 업데이트를 거밋 할 것
- 웹 UI에서는 Finished 상태가 나올 것이며, 클라이언트는 OTA-update1을 실행한 상태

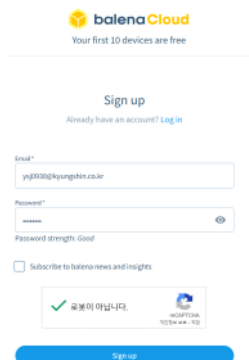
로컬 업데이트 시 balena 사용하기

Balena 는 도커 컨테이너를 사용해 소프트웨어 업데이트를 배포

장치는 balena의 도커 호환 컨테이너 엔진인 balenaengine과 함께 제공되는 Yocto 기반의 리눅스 배포판인 balena OS 실행 OTA 업데이트는 다양한 장치를 관리하기 위한 호스팅 서비스인 balenaCloud 에서 푸시된 릴리스를 통해 자동으로 일어나

• 계정 생성하기

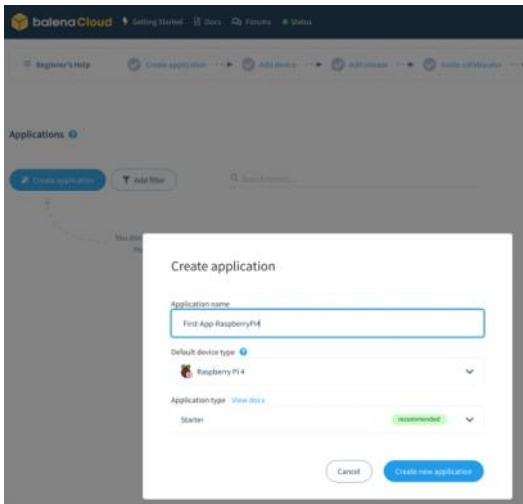
<https://dashboard.balena-cloud.com/signup> 에 접속해 이메일 주소와 비밀번호 입력



The image shows the BalenaCloud sign-up page. At the top, it says 'balenaCloud' and 'Your first 10 devices are free'. Below this is a 'Sign up' button with a link 'Already have an account? Login'. The form has two input fields: 'Email*' with the value 'yng0002@kyungshin.co.kr' and 'Password*'. Below the password field is a 'Password strength: Good' indicator. There is a checkbox for 'Subscribe to balena news and insights'. At the bottom, there is a 'Sign up' button and a small box with a green checkmark and the text '로봇이 아닙니다.' (I am not a robot).

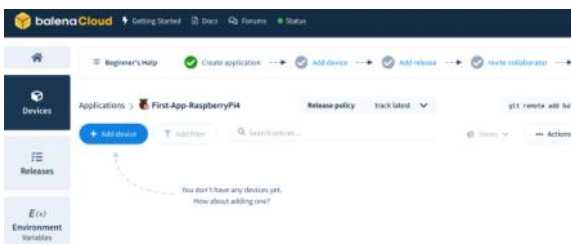
• 애플리케이션 생성하기

balenaCloud 계정에 라즈베리파이 4를 추가하려면 먼저 애플리케이션을 생성

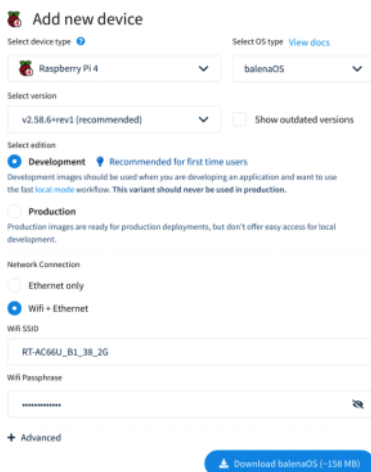


• 장치 추가

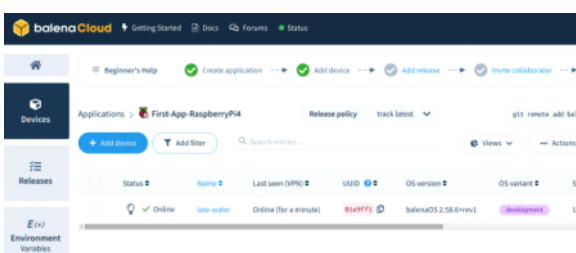
Balena 애플리케이션에 라즈베리 파이 4 추가



- 이메일 주소와 비밀번호로 balenaCloud 대시보드에 로그인
- 우리가 만든 새 애플리케이션을 클릭
- Devices 페이지에서 Add device 버튼 클릭
- 버튼을 클릭하면 Add new device 대화 상자가 나타남
- Raspberrypi 4가 선택한 장치 유형인지 확인
Default device type 으로 Raspberrypi 4를 사용해 애플리 케이션을 생성 했으므로 이 옵션이 이미 선택돼 있어야 함
- balenaOS 가 선택한 OS 인지 확인
- 선택한 balenaOS 버전이 최신 버전인지 확인
- balenaOS 의 에디션으로 Development를 선택
- Network Connection 으로 Wifi_Ethernet 을 선택
- Wi-fi 라우터의 SSID와 암호를 각 항목에 입력



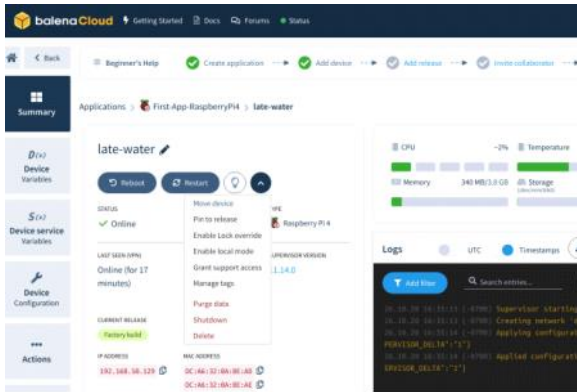
- Download balenaOS 버튼 클릭
- 압축된 이미지 파일을 호스트 시스템에 저장



=> 라즈베리 파이 4를 balena 애플리케이션에 연결 완료

OTA 업데이트를 배포할 수 있도록 로컬 모드 활성화

- balenaCloud 대시 보드의 Devices 페이지에서 타킷 라즈베리 파이4를 클릭
- 라즈베리 파이 4의 장치 대시보드에서 친구 옆에 있는 아래 방향 화살표를 클릭
- 드롭 다운 메뉴에서 Enable local mode 선택



• CLI 설치하기

리눅스 호스트 시스템에 balena CLI 를 설치하는 방법

- 웹 브라우저를 열고 깃허브의 최신 balena CLI 릴리스 페이지로 이동
- 최신 리눅스용 ZIP 파일을 클릭해 다운로드 (balena-cli-Vx.y.z-linux-x64-standalone.zip 형식의 파일명 찾기) X,Y,Z는 메이저,마이너,패치 버전 번호
- 홈 디렉터리에서 ZIP 파일 압축 해제

```
$ cd ~
$ unzip Downloads/balena-cli-v12.25.4-linux-x64-standalone.zip
```

파일 내용물은 balena-cli 디렉터리에 저장

- PATH 환경 변수에 balena-cli 디렉터리를 추가

```
$ export PATH=$PATH:~/balena-cli
```

- 설치 성공 확인

```
$ balena version
12.25.4
```

라즈베리 파이 4의 로컬 네트워크 스캔

```
$ sudo env "PATH=$PATH" bal ena scan
Reporting scan results
-
host:      01e9ff1.local

address:    192.168.50.129
dockerInfo:
  Containers:    1
  ContainersRunning: 1
  ContainersPaused: 0
  ContainersStopped: 0
  Images:        2
  Driver:         overlay2
  SystemTime:    2020-10-26T23:44:44.37360414Z
  KernelVersion: 5.4.58
  OperatingSystem: balenaOS 2.58.6+rev1
  Architecture:  aarch64
dockerVersion:
  Version:    19.03.13-dev
  ApiVersion: 1.40
```

• 프로젝트 푸시하기

로컬 네트워크를 통해 파이썬 프로젝트를 라즈베리 파이로 푸시

- 간단한 "Hello World!" 파이썬 웹 서버 프로젝트를 복제

```
$ git clone https://github.com/balena-io-examples/balena-python-hello-world.git
```

- 프로젝트 디렉터리로 들어가기

```
$ cd balena-python-hello-world
```

- 라즈베리 파이 4로 코드 푸시

```
$ balena push 01e9ff1.local
```

- 도커 이미지가 빌드와 시작을 완료할 때까지 기다린 다음, 애플리케이션이 시작돼 포그라운드에서 동작하고 stdout에 로그를 출력하는

것을 확인

e. 웹 브라우저에서 웹 서버로 요청을 보내기 (현재 나의 장치 IP 주소로 바뀌어야 함)

=> 라즈베리 파이 4에서 실행되는 웹 서버는 'Hello World!'로 응답

```
[Logs] [10/26/2020, 5:26:35 PM] [main] 192.168.50.146 - -  
[27/Oct/2020 00:26:35] "GET / HTTP/1.1" 200 -
```

로그 항목의 IP 주소는 웹 요청을 발행한 시스템의 IP 주소여야 하며, 웹 페이지를 새로 고칠 때 마다 새 로그 항목이 나타나야 함

로그 다시 추적하기

```
$ balena logs 01e9ff1.local
```

01e9ff1.local 을 장치 호스트 이름으로 바꿔 작성해야 함