Putting firmware on the device

a Zephyr+Yocto+Mender hike

Josef Holzmayr
Linux Plumbers Conference - 14.09.2022

About me

Josef Holzmayr

Head of Developer Relations at mender.io

Yocto Project Ambassador
OpenEmbedded Social Media Manager
Gitpod.io Community Hero

Contact Me

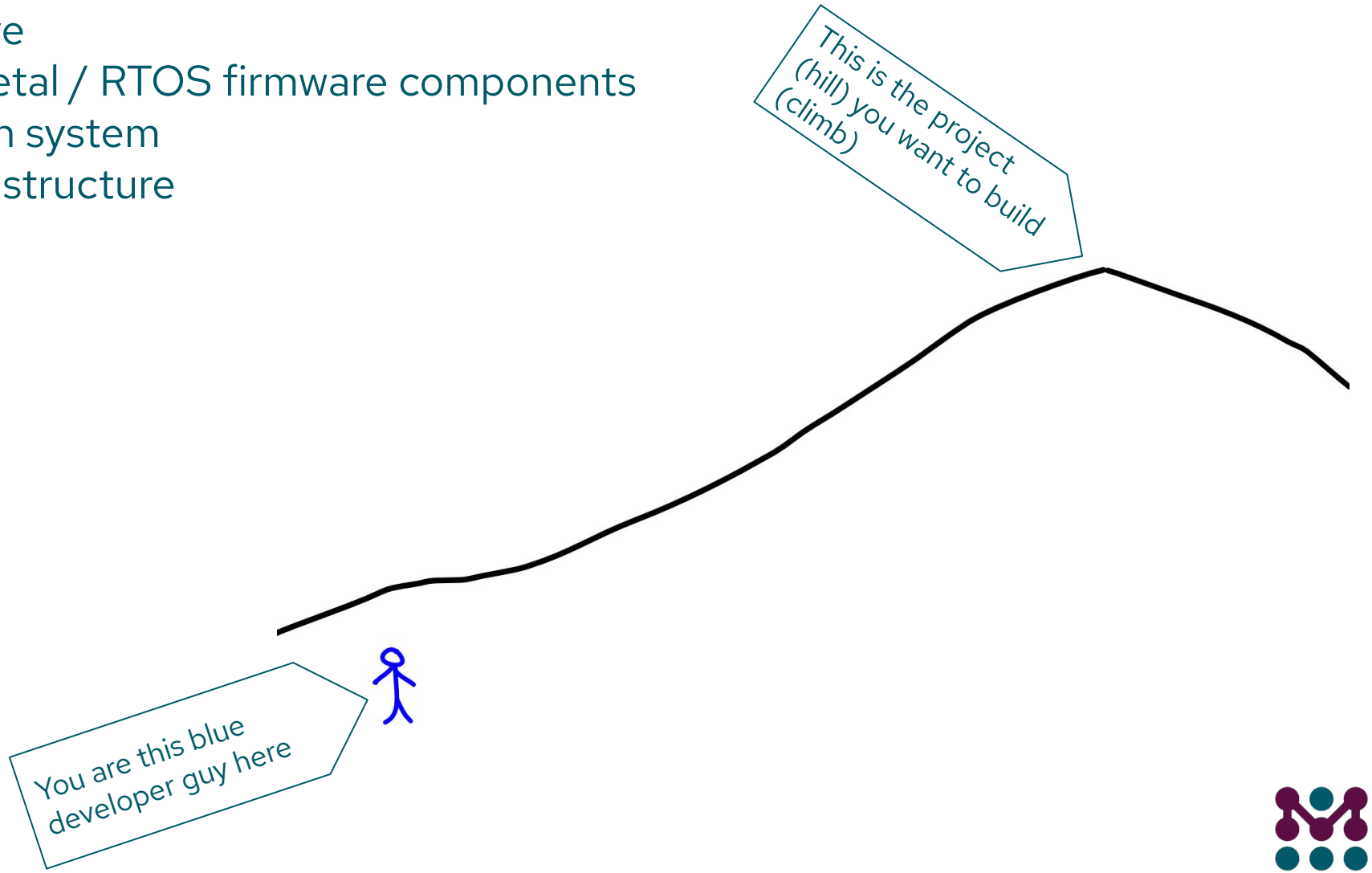
josef.holzmayr@northern.tech

@TheYoctoJester

The hike!

Your project might involve

- a number of bare-metal / RTOS firmware components
- the Linux-bases main system
- middleware and infrastructure
  - connectivity
  - OTA
- application



We will skip for brevity and simplicity

- my usual entertainment parts
- loading the M4 firmware
- communication (RPMsg)
- pushing the image to OTA and deploying it
- the live demo

What is it about? The problem

Firmware development pre-2020ies:

Person A: builds blob for integrated µC A (bare metal / RTOS)
(--> blob-A)

Person B: builds blob for integrated µC B A (bare metal / RTOS)
(--> blob-B)

Person C: builds blob for complete system, integrating µC blobs A and B
(--> blob-C)

Person D: puts blob C through deployment

What is it about? A possible solution

Firmware development DevOps style:

Person A: codes for integrated µC A
(--> source-A)

Person B: codes for integrated µC B
(--> source-B)

Person C: codes for complete system
(--> blob-C)

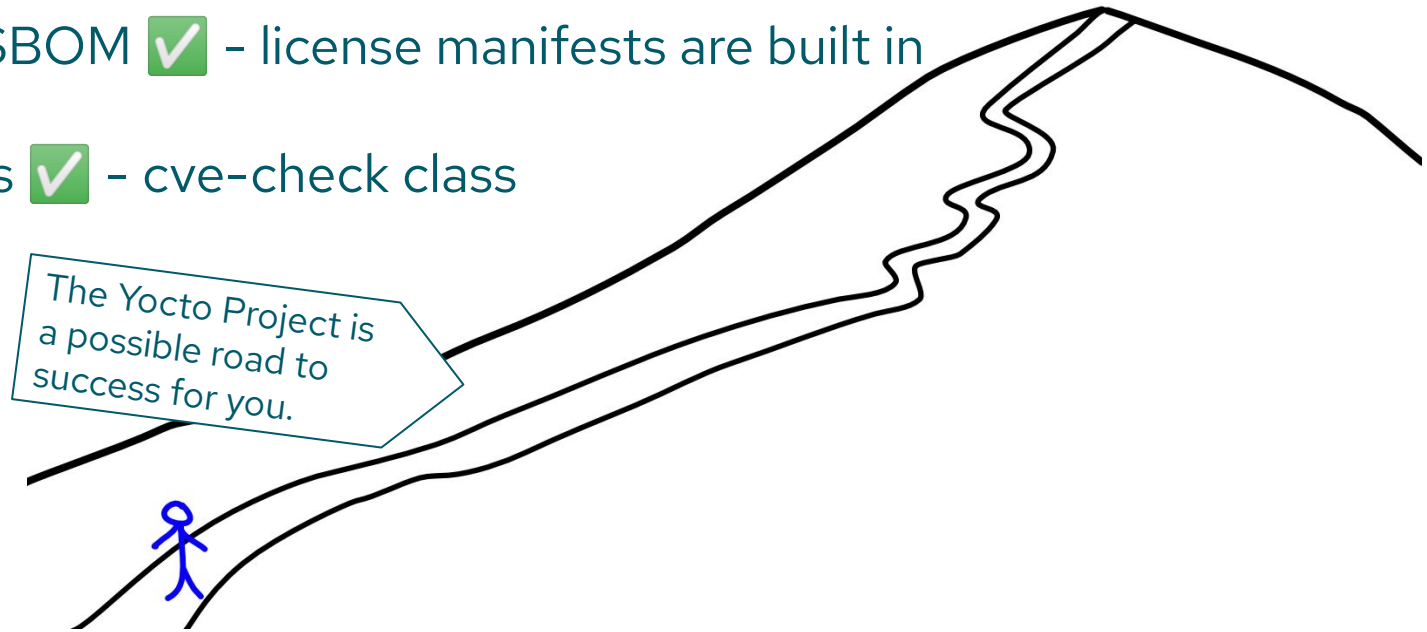
Person D: manages a pipeline that can run coherent builds of A, B and C, as well as push to deployment.

Ds requirements

- Composability
  - builds for several, possibly distinct architectures
  - middleware / infrastructure
- Reproducibility
- Compliance
  - unified License manifests / SBOM
- Security
  - CVE checking all build stages

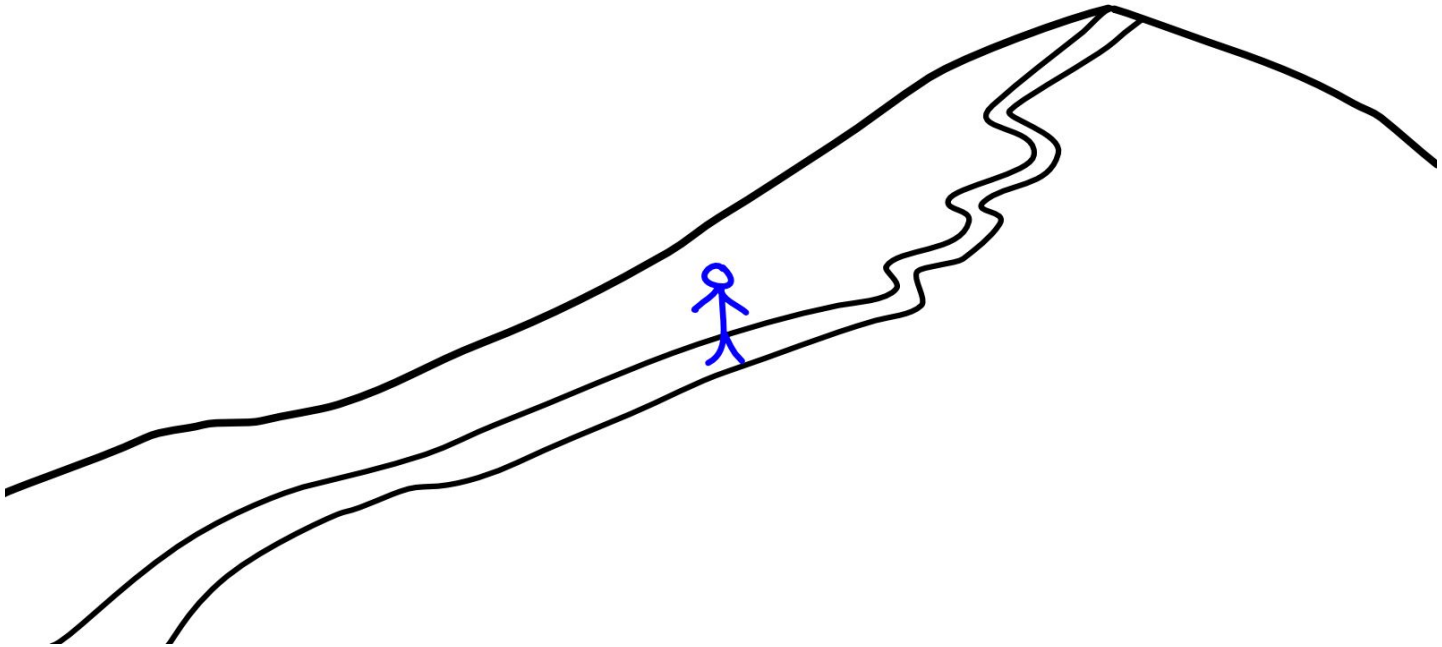
The Yocto Project

- Composability
  - builds for several, possibly distinct architectures ✅ - multiconfig
  - middleware / infrastructure ✅ - layers
- Reproducibility ✅ - building without network, fixed source revisions
- Compliance
  - unified License manifests / SBOM ✅ - license manifests are built in
- Security
  - CVE checking all build stages ✅ - cve-check class



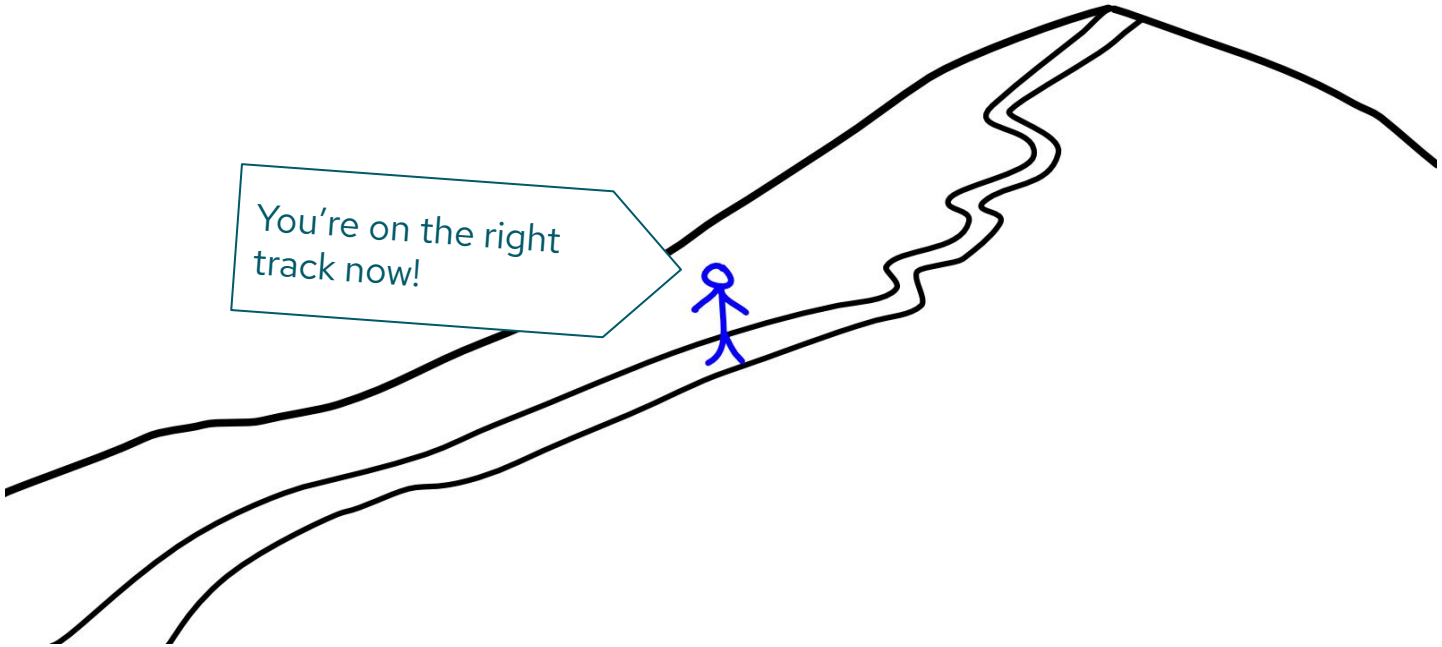
Composing the build 1: Zephyr RTOS

- provides the meta-zephyr layer
- can be integrated as a multiconfig stage
- provides the resulting blob for a later stage to be incorporated



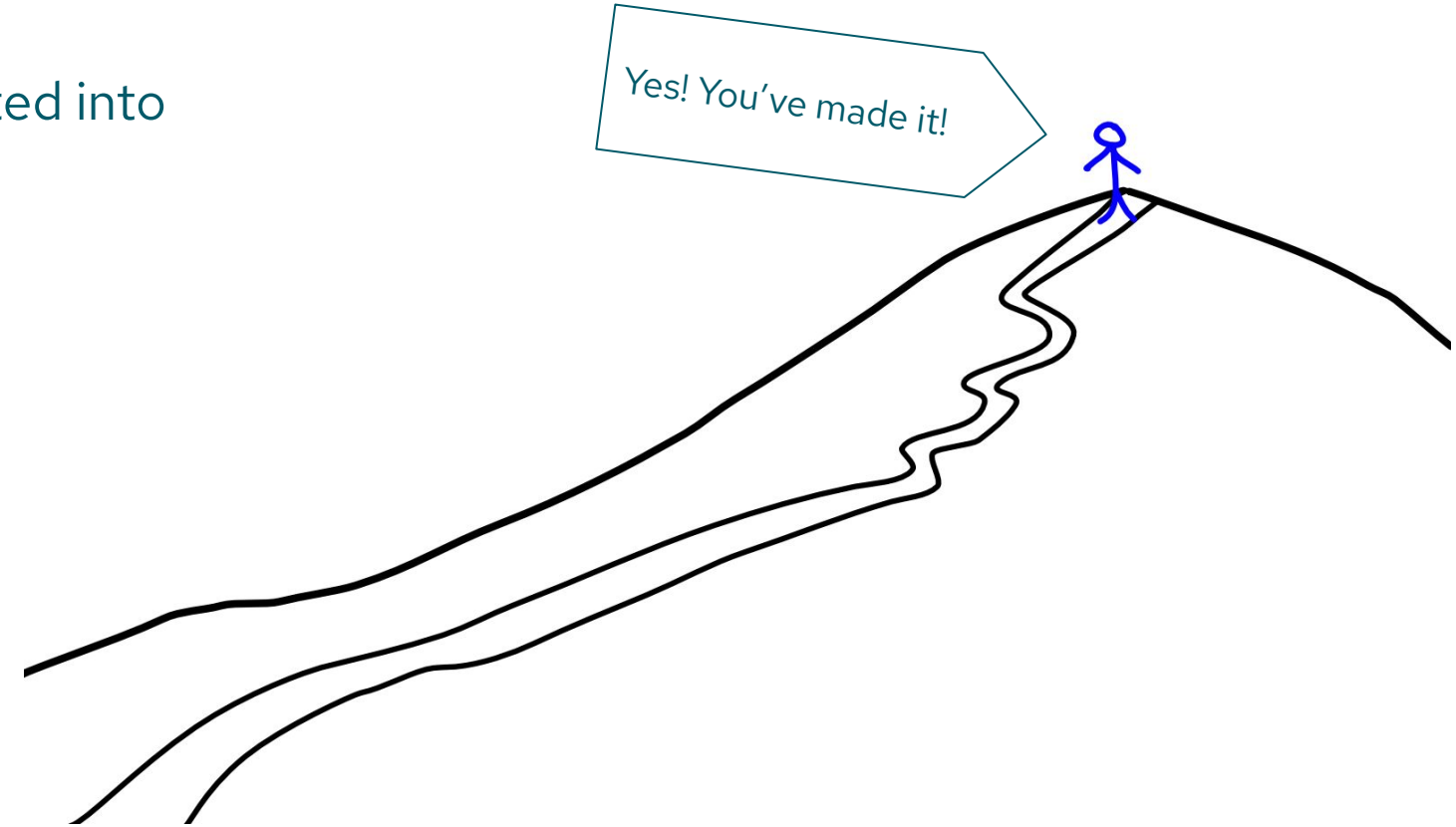
Composing the build 2: Mender

- provides the meta-mender layer
- inert until specifically enabled for a build configuration



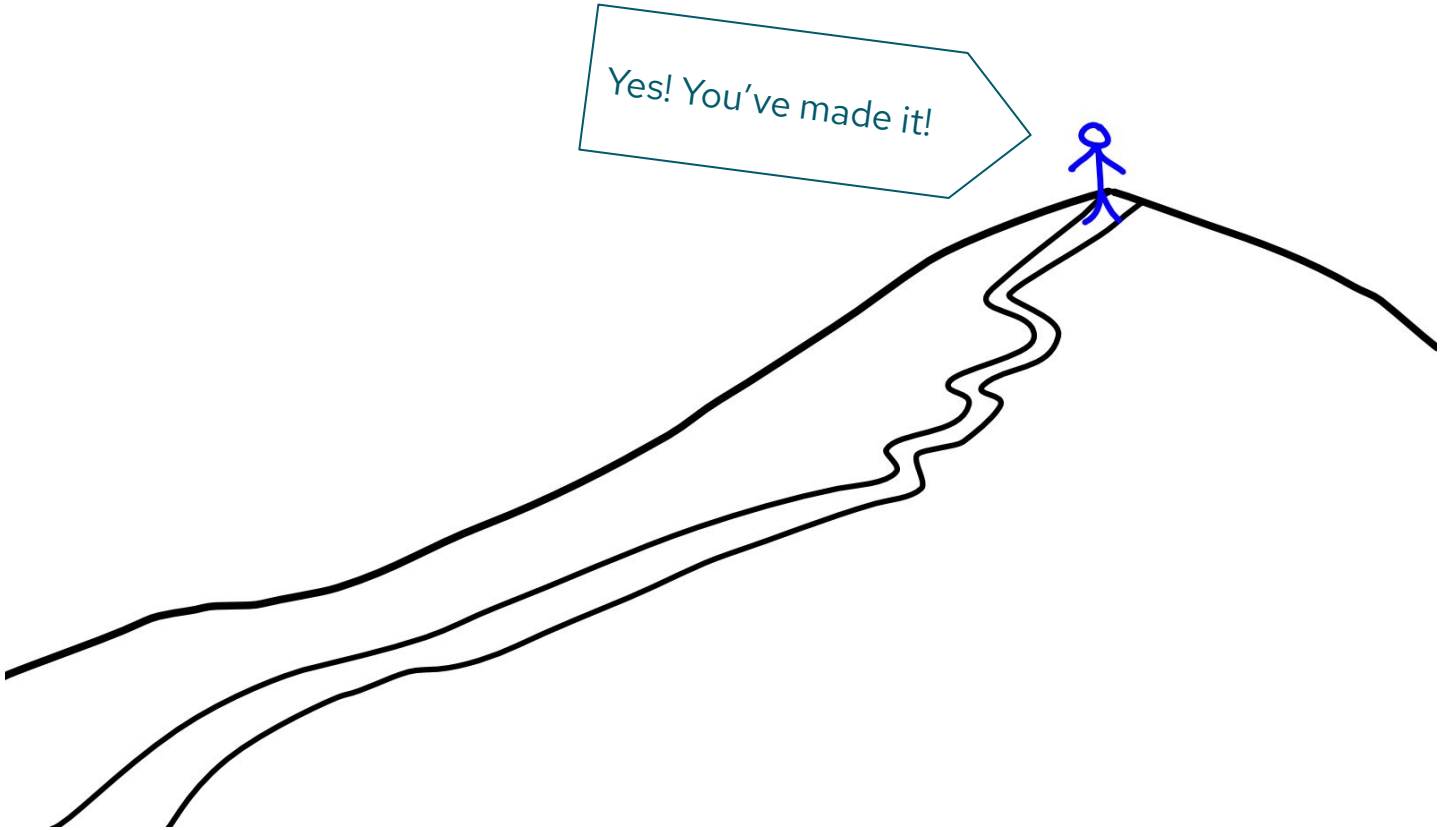
Reproducing the build: a kas file

- Fixing the meta data revisions
- embeddable into the (final) layer
- Defines MACHINE and IMAGE
- arbitrary data that can be injected into local.conf



Reproducing the build: a kas file (excerpt)

```
header:
 version: 11

distro: poky

repos:
 poky:
   url: https://git.yoctoproject.org/git/poky
   refspec: 27fde9cc2aeddb9ea964efb4cc2de7c608256777
   layers:
     meta:
     meta-poky:

local_conf_header:
 base: |
   CONF_VERSION = "2"
   PACKAGE_CLASSES = "package_ipk"
   INIT_MANAGER = "systemd"

target:
 - zirconium-image

machine: colibri-imx7-emmc
```



Learn more

Get started now
docs.mender.io/getting-started

Join the Mender Hub community
hub.mender.io

Mender on Github:
github.com/mendersoftware/

contact@mender.io

mender.io

@mender_io

company/mender.io

Thank You

Q & A