

The impact of your choices on the Yocto Project system you are creating

Yocto Project Dev Day Prague 2026

Marco Cavallini - KOAN

KOAN sas, Bergamo - Italia

October 6, 2026



© 2026 KOAN sas - Licensed under CC BY-SA 4.0.

Embedded Linux consultant and trainer

- Founder of **KOAN** established in 1996, an embedded linux consulting and training company
Based in Bergamo, Italy 🇮🇹
- Embedded Linux consultant (2000)
- Yocto Project participant (2010)
- First european **Yocto Project ambassador** 🇪🇺 (2019)
- Starting from 2025 **KOAN** is part of the **Abinsula group**
- I am also Open Source community advocate at **Abinsula**
- **KOAN** provides the following services:
 - Development, Consultancy, Training for Embedded Linux
 - BSP creation, Board bring-up, Bug fixing, Security and CVE maintenance



Introduction

- All characters appearing in this work are fictitious
- Any resemblance to real persons, living or dead, is purely coincidental
- These are personal opinions simply based on my experience 😊
- Any alternative solution is welcome, and can be discussed in the Q&A session at the end of this talk

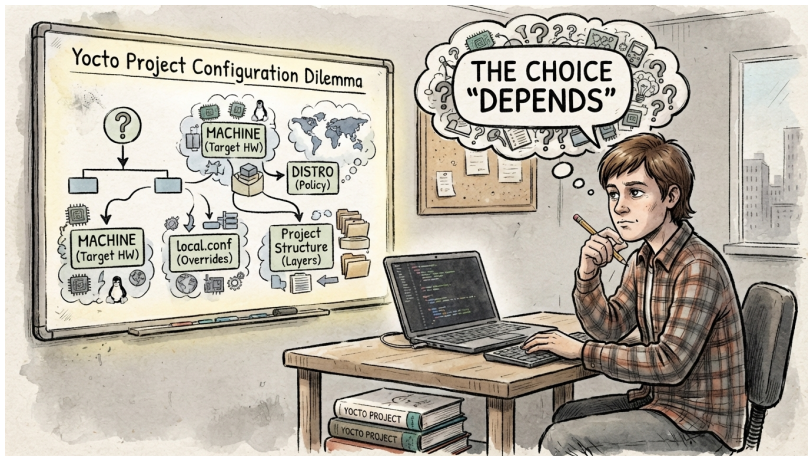


- **The flexibility paradox:** Yocto provides countless ways to achieve identical goals—overriding variables, using `.bbappend` files, writing custom classes, or overriding tasks. Without discipline, this may results in chaotic configurations.
- **Downstream impact:** Choices made early in layer design (e.g., modifying upstream recipes directly vs. inheritance) accumulate technical debt, making security patching and layer updates exponentially harder.
- **Best practices:** Has already been discussed in the Yocto Project community, but looks like they are not reaching an audience wide enough to avoid issues described here.
- The latin phrase “*Repetita juvant*” means that repetition helps learning, and it is the reason why I am repeating some of the best practices in a real context.

- Yocto is a build system based on layers
- Each layer does a specific job and has a specific purpose
- Layers are listed in `BBLAYERS` variable in `bblayers.conf`
- Layers can be combined to create a complete system
- The first rule is **do not modify the underlying layers**
- Instead, create new layers to extend functionality
- The second rule is to keep `local.conf` clean, and avoid too many settings.



What is the best option to achieve the desired result?



- **BSP layer:**
Keep hardware-specific configurations, kernel patches, and bootloaders isolated in dedicated board support layers.
- **Distro layer:**
Define system-wide policies, init managers (systemd/sysvinit), security hardening, and default package selections.
- **Applications layer:**
House proprietary applications, configurations, and product-specific image recipes in a clean top-level layer.
- They may be merged into a single layer for small projects, but **"depends"**.

- **Modifying Upstream Metadata Directly:**
Editing meta-poky or meta-oe instead of using proper bbappends.
- **Overusing `IMAGE_INSTALL:append`:**
Polluting rootfs images without clean package groups (packagegroups).
- **Ignoring License Compliance:**
Failing to track commercial and open-source licenses early in the build pipeline.

- Keep `local.conf` clean, and avoid polluting it with too many `IMAGE_INSTALL:append`
- Manage **software** settings in the `DISTRO` configuration file
- Manage **BSP** settings in the `MACHINE` configuration file
- Collect recipes with same purpose in a **packagegroup**
- **Recipes** are atomic

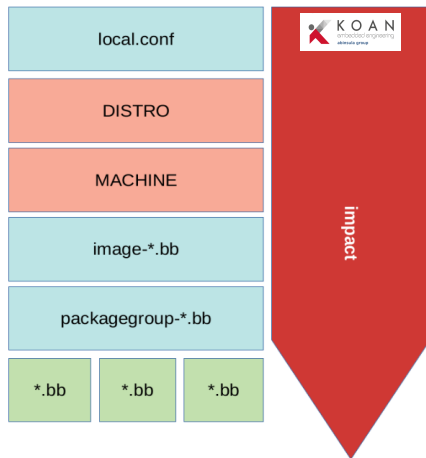
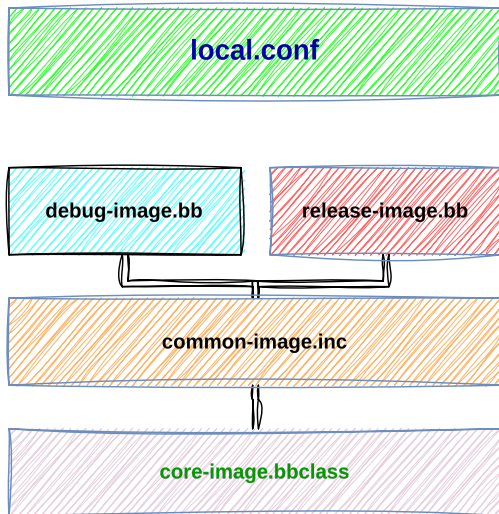


Image source: Koan training slides

Case histories and best practices

Keep `local.conf` clean



Issue: Overusing `IMAGE_INSTALL:append` in `local.conf`.

- *Consequence:* Polluting every rootfs image variant.

Alternative:

- Set `IMAGE_INSTALL:append` in the image recipes only (or `.inc` files).
- In case create a custom `packagegroup` for those components, and add it to the image recipe.

`recipes-core/packagegroups/packagegroup-industrial-connectivity.bb`
(meta-mycompany) 

```
SUMMARY = "Custom packagegroup for industrial connectivity"
PACKAGE_ARCH = "${MACHINE_ARCH}"
inherit packagegroup

RDEPENDS:${PN} = "\
    libgpiod \
    mosquito \
    python3-paho-mqtt \
"
```

Issue: Overusing `CORE_IMAGE_EXTRA_INSTALL` in `local.conf`.

- *Consequence:* Polluting every `core-image-*` image variant.

Alternative: Set `IMAGE_INSTALL:append` in the image recipes only (or `.inc` files).

templates/local.conf

 (meta-company)

```
LICENSE_FLAGS_ACCEPTED = "commercial"
CORE_IMAGE_EXTRA_INSTALL += " memtester htop iotop tmux iperf3 "
CORE_IMAGE_EXTRA_INSTALL += " wvdial "
CORE_IMAGE_EXTRA_INSTALL += " libubootenv-bin u-boot-compulab-env "
CORE_IMAGE_EXTRA_INSTALL += " u-boot-update-script "
CORE_IMAGE_EXTRA_INSTALL += " linux-firmware-ax210 firmware-imx-sdma-imx7d "

CORE_IMAGE_EXTRA_INSTALL:append = " u-boot-script linux-firmware-ax200"
CORE_IMAGE_EXTRA_INSTALL:append = " networkmanager networkmanager-nmcli wireless-tools "
CORE_IMAGE_INSTALL:remove = " connman-client connman-tools "
RDEPENDS:connman-client:remove = "connman"
RDEPENDS:connman-tools:remove = "connman"

CORE_IMAGE_EXTRA_INSTALL:append = " libgpiod-tools eeprom-util bluez5 gpsd modemmanager "
CORE_IMAGE_EXTRA_INSTALL:append = " gps-utils can-utils minicom ntp socat "
CORE_IMAGE_EXTRA_INSTALL:append = " wpa-supPLICANT pulseaudio pulseaudio-server"
```

Multiple MACHINE configurations

- When you have boards with different architectures, it is wise to create a separate layer for each board.
- But with simple board variants (with same architecture), do not create a separate layer for each board.
- Use `MACHINEOVERRIDES` to share common settings between boards with the same SoC or family.
- Use `SOC_FAMILY` to group boards with the same SoC, and share common settings between them.

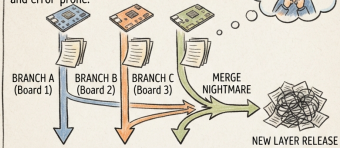
- In this example, each MACHINE settings was managed in a specific branch of the same layer, and had to merge them all together when a new release of the layer was released.
- This was a nightmare to maintain, and it was impossible to keep track of the changes made in each branch.
- The solution could be to use MACHINEOVERRIDES to share common settings between boards with the same SoC.
- This way, each board could be maintained independently in the same layer.

SIMPLIFYING MACHINE SETTINGS WITH MACHINEOVERRIDES

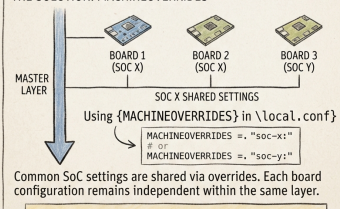
Moving from complex branch merging to a unified layer structure.

THE OLD WAY: BRANCH NIGHTMARE

Maintaining separate branches for each board. Merging all changes on new releases is difficult and error-prone.

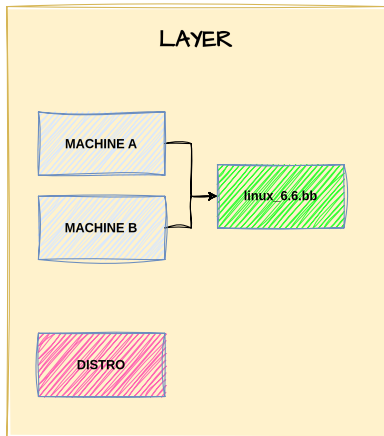


THE SOLUTION: MACHINEOVERRIDES

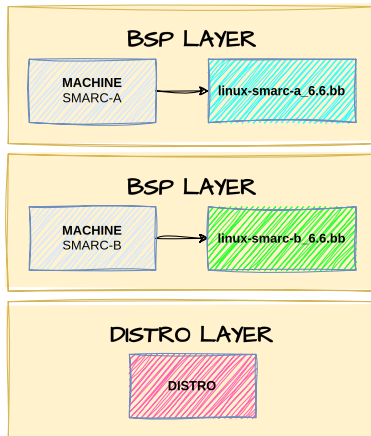


BENEFITS: Easier maintenance, clear history tracking, independent board updates.

Carrier variants on same SoM



SoM variants (SMARC)



Files replacement and patches

Issue: Force file inclusion via `addtask`.

- *Consequence:* Impossible to maintain consistency across different releases.

Alternative: Use patches.

`recipes-kernel/linux/linux-imx.bbappend`

 (meta-mycompany)

```
FILESEXTRAPATHS:prepend := "${THISDIR}/files:"
```

```
SRC_URI += "file://i2c-imx.c"
```

```
do_override {  
    cp ${THISDIR}/i2c-imx.c ${S}/drivers/i2c/  
}
```

```
addtask overrides before do_configure after do_patch
```

Issue: Force file inclusion via `subdir`.

- *Consequence:* Impossible to maintain consistency across different releases.

Alternative: Use patches.

```
recipes-kernel/linux/linux-imx.bbappend
```

 (meta-mycompany)

```
FILESEXTRAPATHS:prepend := "${THISDIR}/files:"
```

```
S = "${WORKDIR}/git"
```

```
SRC_URI += "file://i2c-imx.c;subdir=drivers/i2c"
```

Recipe with conditional file inclusion

Issue: Make the inclusion of the file conditional on the value of `DISTRO`

- Use `SRC_URI` to include the right file based on `DISTRO`
- Use a weird approach in `do_install` to select the right script to install

Alternative: In the next page 

recipes-core/init-setup/init-setup.bb

```
SRC_URI:append = " file://init-setup.sh"
SRC_URI:append:distro-hmi = " file://init-setup-hmi.sh"

do_install () {
    install -d ${D}/${sbindir}
    ...

    if [ -f ${WORKDIR}/init-setup-hmi.sh ]; then
        install -m 755 ${WORKDIR}/init-setup-hmi.sh ${D}/${sbindir}/init-setup.sh
    else
        install -m 755 ${WORKDIR}/init-setup.sh ${D}/${sbindir}/init-setup.sh
    fi
}
```

Issue: From previous page 

Alternative:

- Use `SRC_URI` to include the right file based on `DISTRO`
- Keep the orthodox approach using the task overrides

recipes-core/init-setup/init-setup.bb

```
SRC_URI:append:distro-normal = " file://init-setup.sh"
SRC_URI:append:distro-hmi = " file://init-setup-hmi.sh"

do_install () {
    install -d ${D}/${sbindir}
    ...
}

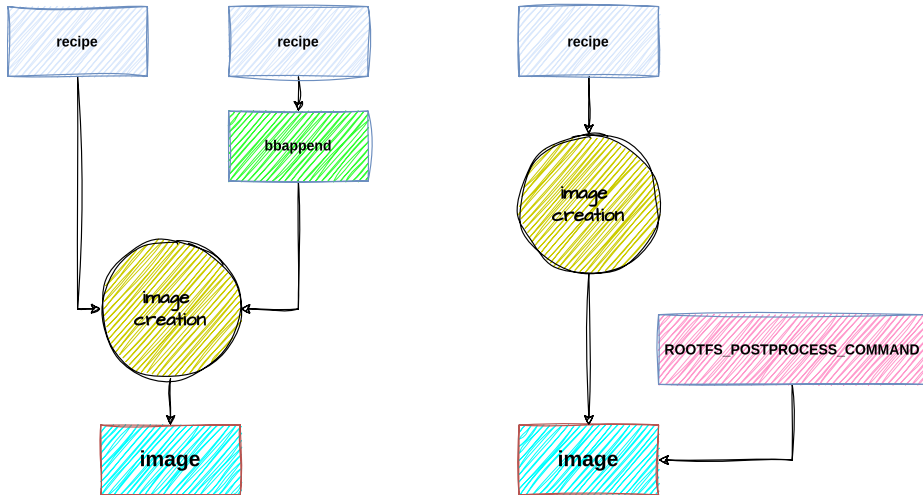
do_install:append:distro-normal () {
    install -m 755 ${WORKDIR}/init-setup.sh ${D}/${sbindir}/init-setup.sh
}

do_install:append:distro-hmi () {
    install -m 755 ${WORKDIR}/init-setup-hmi.sh ${D}/${sbindir}/init-setup.sh
}
```

Modify a configuration file

1/4 - Modify a configuration file

Two different ways to modify a configuration file in the rootfs image



- In this case `ROOTFS_POSTPROCESS_COMMAND` looks a faster solution
- Customizing the recipe responsible for that setting is a more orthodox solution
- The first solution is faster, but the second is more atomic and easier to maintain

Issue: Modify a configuration file using `ROOTFS_POSTPROCESS_COMMAND` in the image recipe

Alternative: In the next page →

recipes-images/images/custom-image.bb

```
# set static IP (only in mainos)
set_static_ip () {
    cat << EOF > ${IMAGE_ROOTFS}${sysconfdir}/network/interfaces

# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)
auto lo
iface lo inet loopback

iface eth0 inet static
    address 192.168.0.2
    netmask 255.255.255.0
    network 192.168.0.0
    gateway 192.168.0.1
EOF

ROOTFS_POSTPROCESS_COMMAND += "set_static_ip;"
```

Issue: From previous page 

Alternative:

- Customize your `/etc/network/interfaces` file in `.bbappend`

`recipes-core/init-ifupdown/init-ifupdown_%.bbappend`

```
# Gets the new file from the custom layer instead of the original in meta  
FILESEXTRAPATHS:prepend := "${THISDIR}/files:"
```

interfaces

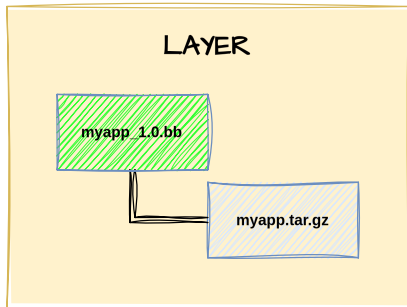
```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)  
auto lo  
iface lo inet loopback  
  
iface eth0 inet static  
    address 192.168.0.2  
    netmask 255.255.255.0  
    network 192.168.0.0  
    gateway 192.168.0.1
```

Installing files from a tarball

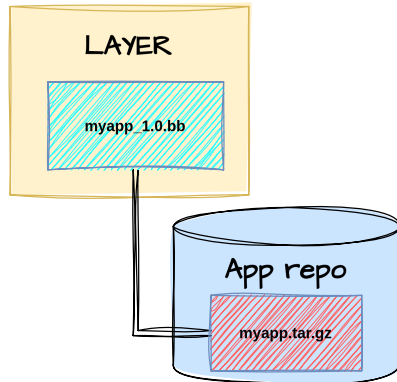
- Sometimes, you need to include files from a tarball in the root filesystem.
- As a best practice, avoid storing such tarballs directly in the recipe or layer repository. Every time the tarball changes, a new version is committed to the Git repository, which can quickly make the layer unnecessarily large.
- I have seen customer-specific meta-layers grow to more than 1 GB simply because large tarballs were committed to the repository.
- A better solution is to store the tarball in a separate location and fetch it from an external source during the build.

Two different ways to install files from a tarball in the rootfs image

Tarball into a recipe layer



Tarball out of Yocto tree



Issue:

- Install a whole set of files from a tarball local in the layer.
- The tarball will be committed in the layer repository.

Alternative: In the next page 

meta-custom/recipes-apps/myapp/myapp_1.0.bb

```
SRC_URI = " \
    file://${PN}.tar.gz \
    file://start.sh \
    "

do_install () {
    install -d ${D}/opt/customer
    cp -a ${WORKDIR}/${PN} ${D}/opt/customer
    install -m 755 ${WORKDIR}/start.sh ${D}/opt/customer/${PN}/bin
}
```

Issue: From previous page 

Alternative:

- Install a whole set of files from a tarball out of Yocto tree.
- The tarball can be managed in an independent way.

meta-custom/recipes-apps/myapp/myapp_1.0.bb

```
SRC_URI = " \
    https://somewhere.net/project/${PN}-${PV}.tar.gz \
    file://start.sh \
    "

do_install () {
    install -d ${D}/opt/customer
    cp -a ${WORKDIR}/${PN} ${D}/opt/customer
    install -m 755 ${WORKDIR}/start.sh ${D}/opt/customer/${PN}/bin
}
```

Wrapping Up

- Customization of a Yocto Project system is a long term commitment, and it is important to make the right choices from the beginning.
- Most of we discussed here are common sense, but they are often misunderstood in practice.
- Keep your `local.conf` clean, and avoid polluting it with too many settings.
- Use `MACHINEOVERRIDES` to share common settings between boards with the same SoC or family.
- Keep your modification atomic; a recipe manages a component, and it should be easy to understand what it does.
- Don't relay verbatim on what a vendor layer does, but understand it and make your own choices.



Questions? Comments?

- My email: m.cavallini@koansoftware.com
- Sources (L^AT_EX):
<https://gitlab.com/koansoftware/<title>>
- Credits:
 - Thanks to Michael for his LaTeX template used to generate these slides 😊
 - Where not specified images are generated with Gemini AI
- Slides available under the CC-BY-SA 4.0 license
<https://koansoftware.com/pub/talks/>

