amarula solutions
Go Far, Go Together

TRAINING PROGRAMME

FIVE-DAY HANDS-ON TRAINING

Yocto Project

From the first build to a reproducible release

A five-day, hands-on course for embedded Linux teams: OpenEmbedded-Core and BitBake, recipes and layers, images and WIC, kernel and U-Boot, devtool and the SDK, licence compliance, and reproducible KAS builds.

DURATION	TRAINER	MINIMUM	PRICE
5 days (4 core + capstone)	**Michael Trimarchi**	**3 participants**	**€1,500 per participant**

Delivered on site only. The course is not available online.

OVERVIEW

From first build to reproducible release

Yocto is the build system behind many embedded Linux products. This five-day, hands-on course takes a team from the first build to a reproducible product release: the metadata model behind BitBake, recipes and layers, images and WIC, kernel and U-Boot, devtool and the SDK, licence compliance, and KAS projects that build the same way on a laptop, in a container and in CI.

FORMAT	LABS	AUDIENCE	MINIMUM
Instructor-led, hands-on	**KAS, QEMU and i.MX8M Plus**	**Embedded Linux teams**	**3 participants**

WHO IT IS FOR

- BSP and platform engineers responsible for machines, kernels and bootloaders.
- Application and integration teams packaging software and using the SDK.
- Teams moving from Buildroot, vendor SDKs or ad-hoc build scripts.

PREREQUISITES

- Linux command line and Git.
- C programming.
- Embedded Linux basics: bootloader, kernel and root filesystem.
- A Linux host for the labs (Ubuntu 24.04 recommended).

WHAT YOU WILL LEARN

- ✓ Build and boot a Yocto image with KAS.
- ✓ Read and write recipes, classes and `.bbappend` files.
- ✓ Create a layer and a BSP machine.
- ✓ Customise images with packages, WIC and kernel fragments.
- ✓ Use devtool and the SDK in a day-to-day development workflow.
- ✓ Produce licence and SBOM reports and a reproducible release.

On-site delivery only. The course is not available online, and sessions run for at least three participants. The programme can be tailored to your board and product.

PROGRAMME

The five-day programme

The suggested timing is five days: four core days plus one day for the capstone and advanced modules. Each block ends with a checkpoint and a lab.

DAY 1 — Foundations

- Linux boot: bootloader, kernel and root filesystem
- Android, Buildroot and Yocto: choosing a distribution
- BitBake, metadata, OpenEmbedded-Core and layers
- Build environment, directory structure and configuration
- First image built with KAS and booted in QEMU

DAY 2 — Metadata

- Variables, operators, overrides and `bitbake -e`
- Parsing, task graph, dependencies and scheduling
- Recipe structure, `SRC_URI`, patches and packaging
- Classes, `.bbappend` files, layers, machines and distros

DAY 3 — Images and BSP

- Image recipes, packagegroups and `IMAGE_FEATURES`
- `IMAGE_FSTYPES`, WIC and `.wks` disk layouts
- Kernel configuration fragments, device tree and modules
- U-Boot configuration flow, FIT images, signing and secure boot

DAY 4 — Development and releases

- devtool and recipetool; standard and extensible SDK
- systemd services, users, alternatives and package management
- Debugging build and boot failures; sstate and build performance
- Licence compliance, SBOM/SPDX and CVE checks
- KAS projects, lockfiles and reproducible CI builds

DAY 5 — Capstone and advanced modules

- Capstone: a custom recipe, layer, systemd service, kernel fragment, WIC image, SDK and a locked KAS release
- Advanced: multiconfig and multilib, sstate mirrors, secure boot and OTA updates
- CI automation, SBOM/CVE reporting, checkpoints and Q&A

LEARN

- Short lectures and live demonstrations.
- Architecture and trade-offs explained in context.
- A checkpoint and discussion after each block.

BUILD

- A KAS lab after each block.
- Build and boot images under QEMU.
- Debug with devtool and complete a capstone image.

Selected slides from the course deck are reproduced on the following pages. The complete deck, lab exercises and recipes are in the public course repository.

HANDS-ON LABS

Build, boot and inspect a real image

The labs use KAS to fetch OpenEmbedded-Core, BitBake, meta-openembedded and the training layer, generate the configuration and build `training-image`: `core-image-minimal` plus a small C application, a target information script, SSH, a kernel fragment and a WIC image.

```
$ cd labs
$ ./kas.sh build kas/qemu.yml
$ ./kas.sh shell kas/qemu.yml -c 'runqemu qemux86-64 training-image'

# From the target serial console
root@qemux86-64:~# hello
Hello from the Yocto training image!
root@qemux86-64:~# training-info
=== Yocto training image ===
Linux qemux86-64 ...
```

QEMU

- The default machine is `qemux86-64`.
- Students build the image, boot it with `runqemu`, run the training applications and inspect the kernel configuration.

ENGICAM I.CORE MX8M PLUS

- A separate KAS configuration targets the i.MX8M Plus board when the customer BSP layer is available.
- The same workflow carries over from the lab to the product.

WHAT THE EXERCISES COVER

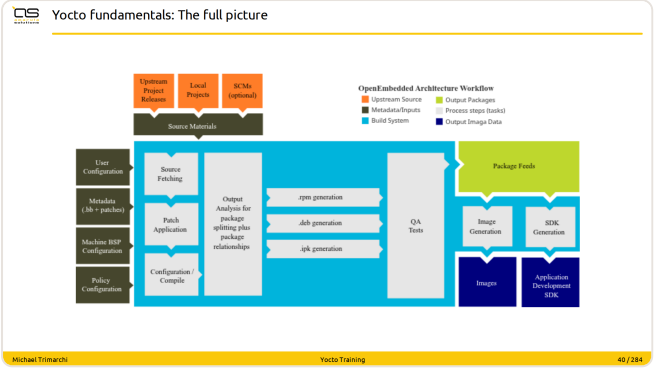
- ✓ Inspect the KAS-resolved configuration and the generated `local.conf` and `bblayers.conf`
- ✓ Build the image and boot it in QEMU
- ✓ Explore the build directory, metadata and task graph
- ✓ Modify and finish a recipe with devtool
- ✓ Customise the image and its WIC disk layout
- ✓ Add a kernel configuration fragment
- ✓ Build and use the SDK
- ✓ Add a systemd service
- ✓ Debug build and runtime failures
- ✓ Inspect shared state and build performance
- ✓ Complete the capstone image

> **Lab requirements:** a Linux host (Ubuntu 24.04 recommended), Python 3.9+, `pipx` or a Python virtual environment, Git, 50 GB of free disk space, 8 GB of RAM and network access for the first build. On Ubuntu 24.04 and later, enable unprivileged user namespaces or use `kas-container`.

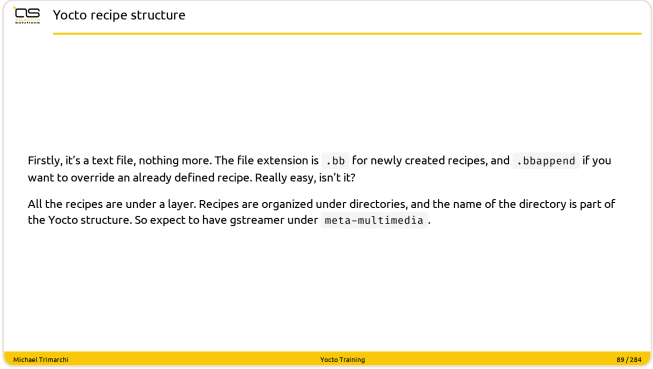
INSIDE THE COURSE

Selected slides

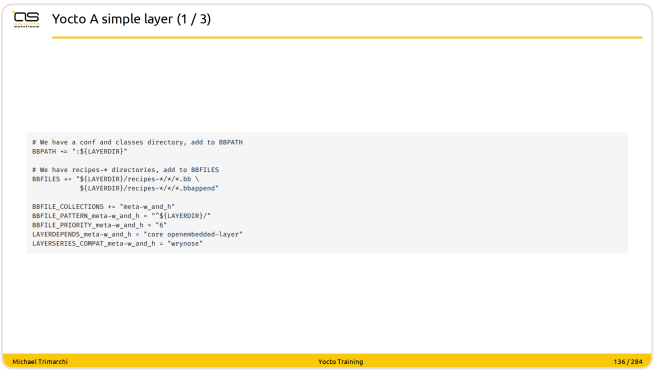
A few pages from the deck: the OpenEmbedded architecture workflow, recipes and layers, WIC images, kernel configuration and the KAS project model.



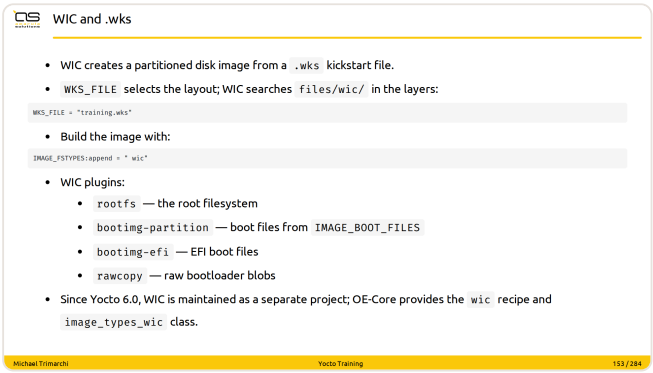
The full picture. OpenEmbedded architecture workflow, from source materials to images and the SDK.



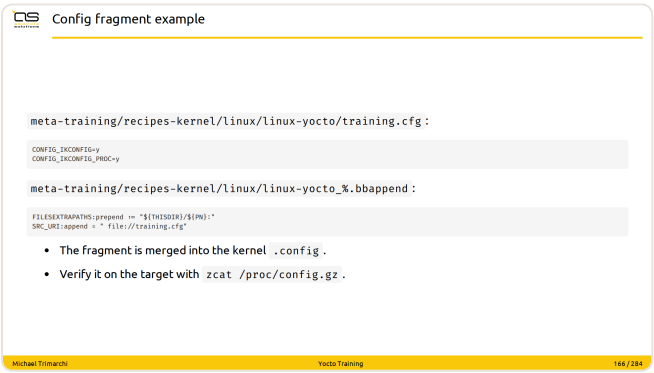
Recipe structure. A `.bb` file describes how to fetch, patch, build and package a component.



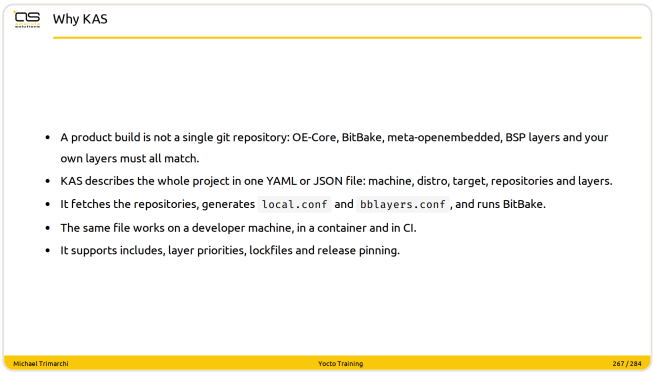
A simple layer. `conf/layer.conf`, `BBPATH` and `BBFILES` in a new layer.



WIC and `.wks`. Building partitioned disk images from a kickstart layout.



Kernel fragment. A configuration fragment and a `.bbappend` enable `/proc/config.gz`.



Why KAS. One YAML project file for OE-Core, BSP layers, lockfiles and CI.

> The complete deck, lab exercises and recipes are in the **yocto-training** repository at github.com/amarula/yocto-training, licensed under CC BY-SA 3.0.

TRAINER AND BOOKING

Your trainer



Michael Trimarchi

Co-Founder & CEO, Amarula Solutions

Michael is a Linux software engineer working on embedded and real-time systems. He holds a Master's degree in Software Engineering from the University of Pisa and is co-author of the first version of the EDF patches submitted to mainline Linux.

Michael teaches this training from the perspective of teams that ship Yocto-based products: the metadata and build decisions that keep a BSP maintainable across releases, and the release process that makes a build reproducible.

Book the course

DURATION	PARTICIPANTS	PRICE	TRAINER
5 days	**Minimum 3**	**€1,500 per participant**	**Michael Trimarchi**

On-site delivery only. Sessions are arranged on request; tell us about your board and preferred dates.

Request a quotation

Use the quotation form at amarulasolutions.com/quotation/ to tell us about your team and preferred dates.

Talk to us

Contact the Amarula team at amarulasolutions.com/contact/ or info@amarulasolutions.com.

© 2026 Amarula Solutions. Selected slides are excerpts from the course deck and remain the property of Amarula Solutions.