Faysal Aziz

Embedded Linux in Practice

From Board Bring-Up to Custom Distributions

FA-0185 · 3 days

DevOps, Cloud & Infrastructure

Introduction

Build, customize, and deploy real embedded Linux systems the way industry actually does it.

Duration: 3 days

Embedded Linux is no longer a niche skill reserved for a handful of specialists; it is the backbone of modern products spanning consumer electronics, industrial automation, networking equipment, medical devices, and automotive systems. What separates a functional embedded Linux engineer from someone who merely "knows Linux" is the ability to shape the operating system itself, bootloaders, kernels, root filesystems, and build systems, to meet real hardware and product constraints.

This intensive three-day course is designed to bridge that gap. It focuses on how embedded Linux systems are actually built and deployed in industry, using widely adopted hardware platforms and toolchains. The emphasis is practical, systematic, and grounded in real workflows rather than academic abstractions.

The instructor brings over 30 years of industry experience and teaches using production-relevant practices, tooling, and problem-solving approaches that are demanded in real embedded Linux roles today.

Learning outcomes

By the end of this course, participants will be able to:

- Understand the complete embedded Linux software stack and how its components interact
- Confidently work with bootloaders, the Linux kernel, and root filesystems on real hardware
- Build and customize embedded Linux systems using Yocto Project workflows
- Perform board bring-up tasks on BeagleBone Black and Raspberry Pi class hardware
- Configure, cross-compile, and deploy embedded Linux software from scratch

- Debug boot, kernel, and user-space issues using industry-standard techniques
- Design reproducible, scalable embedded Linux build environments suitable for products

---

Prerequisites (Non-Negotiable)

Participants **must** meet all of the following prerequisites before attending:

Solid working knowledge of Linux from the command line

- File system navigation
- Permissions and ownership
- Shell usage and basic scripting

Prior exposure to C programming

- Understanding of compilation and linking
- Comfort reading and modifying C source code

Basic understanding of embedded systems concepts

- CPU, memory, storage, and peripherals
- Difference between bare-metal and OS-based systems

Familiarity with software build concepts

- Makefiles and build workflows (conceptual understanding is sufficient)

Participants who do not meet these prerequisites will struggle to keep pace in an intensive environment.

Mandatory Hardware, Software, and Environment Requirements

These requirements are **strict and non-compromisable**:

Host System Requirements

- A dedicated laptop or workstation (no shared systems)

  Ubuntu Linux installed as the primary OS

  - Native installation preferred (no dual-boot time constraints)
- Administrator (sudo) access on the training machine
- Minimum 16 GB RAM (32 GB strongly recommended for Yocto builds)

- Minimum 200 GB of free disk space

  Unrestricted, high-speed internet access
  - No firewalls, proxies, or blocked ports
  - Required for source downloads and build dependencies

Target Hardware Requirements

- BeagleBone Black (mandatory)
- Raspberry Pi (any recent model with sufficient RAM)
- MicroSD cards (minimum 16 GB, Class 10 recommended)
- USB-to-Serial cable (3.3V compatible)
- Ethernet cables and/or supported Wi-Fi adapters
- Reliable power supplies for all target boards

Failure to meet any of the above requirements will limit participation and hands-on effectiveness.

Training outline

1. Embedded Linux Foundations

1. What makes Linux “embedded”
2. Differences between desktop Linux and embedded Linux systems
3. Typical embedded Linux system architecture
4. Responsibilities of the embedded Linux engineer
5. Overview of common embedded Linux use cases and product constraints
6. Lifecycle of an embedded Linux product

---

2. Embedded Linux Software Stack Overview

1. Boot ROM and first-stage boot concepts
2. Bootloader responsibilities and execution flow
3. Linux kernel role and responsibilities
4. Root filesystem structure and purpose
5. User space initialization and system services
6. Interaction between hardware, kernel, and user space

---

3. Host Development Environment Setup

1. Ubuntu as a development host for embedded Linux
2. Required development packages and tools
3. Directory organization for embedded Linux projects
4. Cross-compilation concepts and terminology
5. Native vs cross toolchains
6. Managing multiple toolchains on a single host

---

4. Introduction to Cross-Compilation

1. Why cross-compilation is required
2. Target architecture vs host architecture
3. Cross-compiler structure
4. Sysroot concepts
5. Common cross-compilation pitfalls
6. Verifying cross-compiled binaries

---

5. Bootloaders in Embedded Linux

1. Purpose of a bootloader in embedded systems
2. Boot sequence from power-on to kernel execution
3. Overview of U-Boot architecture
4. U-Boot source tree structure
5. Board configuration and initialization flow
6. Environment variables and persistence
7. Boot commands and scripting
8. Loading kernel images and device trees
9. Booting from different storage media
10. Serial console usage and diagnostics

---

6. Linux Kernel for Embedded Systems

1. Role of the Linux kernel in embedded devices
2. Kernel source tree layout

3. Kernel configuration system
4. Understanding defconfig and board-specific configs
5. Device drivers and kernel subsystems
6. Device tree concepts and structure
7. Device tree source files and compilation
8. Kernel image formats used in embedded systems
9. Kernel build process
10. Kernel deployment to target hardware

---

7. Kernel Configuration and Customization

1. Menuconfig and configuration workflows
2. Enabling and disabling kernel features
3. Selecting CPU architecture options
4. Memory management considerations
5. Power management options
6. File system support
7. Networking stack configuration
8. Peripheral and bus support
9. Debugging and tracing options

---

8. Root Filesystem Concepts

1. Purpose of the root filesystem
2. Directory structure and standards
3. Static vs dynamic root filesystems
4. Init systems overview
5. BusyBox fundamentals
6. Shared libraries and dynamic linking
7. Managing dependencies in embedded systems

---

9. Building a Root Filesystem

1. Creating minimal root filesystems
2. Integrating BusyBox

3. Adding system utilities
4. Managing configuration files
5. Startup scripts and initialization
6. Testing root filesystem integrity
7. Deploying root filesystems to target hardware

---

10. Introduction to Yocto Project

1. Why Yocto is used in industry
2. Yocto terminology and concepts
3. Poky reference distribution overview
4. Yocto workflow and build philosophy
5. Advantages over manual build systems
6. Typical Yocto-based product workflow

---

11. Yocto Build System Architecture

1. BitBake fundamentals
2. Metadata structure
3. Layers and layer priorities
4. Recipes and append files
5. Configuration files and variables
6. Machine configuration concepts
7. Distribution configuration concepts

---

12. Setting Up a Yocto Build Environment

1. Host system requirements for Yocto
2. Directory structure and workspace setup
3. Initializing Yocto build environment
4. Selecting target machines
5. Understanding build output artifacts
6. Managing downloads and shared state cache

---

13. Working with Yocto Recipes

1. Recipe structure and syntax
2. Fetching source code
3. Patching source code
4. Configuring build steps
5. Compile, install, and package tasks
6. Managing dependencies between recipes
7. License handling and compliance concepts

---

14. Customizing Images with Yocto

1. Image recipes and image types
2. Selecting packages for an image
3. Adding and removing features
4. Root filesystem customization
5. Including kernel and bootloader artifacts
6. Image generation and deployment formats

---

15. Yocto for BeagleBone Black

1. Machine configuration for BeagleBone Black
2. Bootloader integration
3. Kernel selection and customization
4. Device tree handling
5. Image deployment workflow
6. Booting Yocto images on hardware

---

16. Yocto for Raspberry Pi

1. Raspberry Pi machine configurations
2. Firmware and bootloader considerations
3. Kernel integration specifics
4. Storage layout and boot flow
5. Network and peripheral enablement

6. Boot validation and troubleshooting

---

17. System Bring-Up and Debugging

1. Serial console debugging techniques
2. Bootloader troubleshooting
3. Kernel boot log analysis
4. Root filesystem mount issues
5. Init and startup failure diagnosis
6. Common hardware-software integration issues

---

18. Embedded Linux Networking Basics

1. Network interface configuration
2. Ethernet bring-up
3. Basic Wi-Fi concepts
4. Network services and testing
5. Remote access and file transfer

---

19. Application Deployment in Embedded Linux

1. Deploying custom applications
2. Cross-compiling user-space programs
3. Integrating applications into Yocto images
4. Startup integration
5. Managing runtime dependencies

---

20. Embedded Linux Best Practices

1. Reproducible builds
2. Version control strategies
3. Managing board support packages
4. Long-term maintenance considerations
5. Debug vs production configurations
6. Preparing systems for field deployment

21. Industry Perspective and Real-World Workflows

1. Typical embedded Linux project structure
2. Interaction between hardware, firmware, and software teams
3. Common mistakes made by new embedded Linux engineers
4. Skills expected in embedded Linux job roles
5. Transitioning from learning to production systems

Disclaimer

This course outline is a guideline. The instructor reserves the right to modify, reorder, expand, or reduce content at any time based on participant preparedness, absorption capacity, available hardware, environmental constraints, and real-time training dynamics. The goal is effective learning and practical skill transfer rather than rigid adherence to a predefined structure.

© 2026 Faysal Aziz · Kuala Lumpur