Building a multi-arch CI pipeline for 13 targets. What could possibly go wrong?

Marek Pikuła · FOSDEM 2026 · Brussels · 2026-02-01

Samsung R&D Institute Poland

→ Introduction

What it is?

Hot takes

What's next?

INTRODUCTION

Who am I?

①

FPGA Gateware

Experience in developing gateware for specialized equipment involving fast interfaces and soft cores.

②

Platform Software

Tizen OS platform software developer focused on board support, boot-chain, kernel, and system libraries for the RISC-V ecosystem as part of RISE.

③

CI Workflows

Recently, a lot of effort into multi-platform CI setups for software projects.

INTRODUCTION

The maintainer problem

① Cross-compile is not cross-test

Cross-compiling proves your code can be built for a target, but only running the test suite on that target tells you whether it actually behaves correctly.

② Maintainers need confidence

Maintainers merge changes faster when CI gives them repeatable, multi-architecture signal instead of relying on a contributor's local setup and promises.

③ Hardware scarcity blocks upstream acceptance

When the only way to validate a platform is scarce hardware or bespoke runners, multi-arch support becomes slow, fragile, and easy to reject upstream.

Let's try to make multi-arch CI as boring as possible

The **ci-multiplatform** project turns cross-platform testing into a routine MR check — using **reusable GitLab CI templates**, **layered OCI images**, and **QEMU user-mode** on semi-unprivileged runners. It is designed so upstream projects can add **coverage across many targets** without building a custom CI snowflake.

Introduction

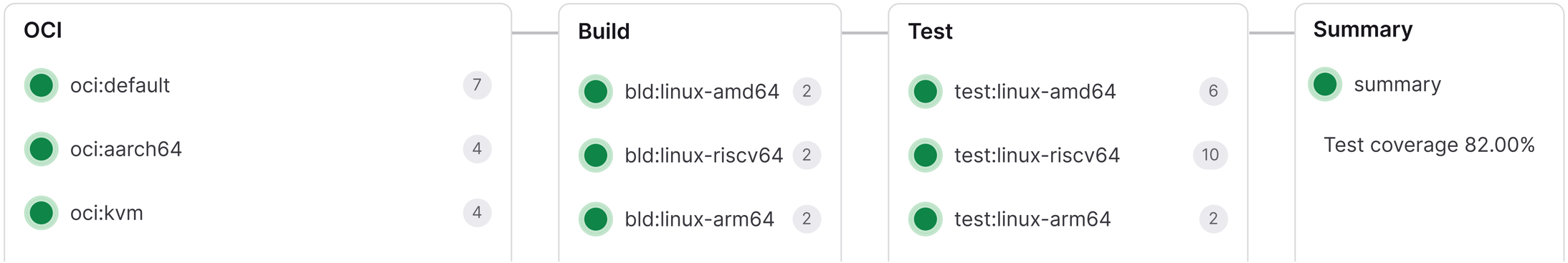
→ What it is?

Hot takes

What's next?

WHAT IT IS?

Building blocks



①

Layered OCI images

Base GNU + LLVM + Meson/CMake/ Autotools + Project dependencies

②

Reusable templates

Component-style includes for easier reuse and customization.

③

QEMU user-mode

Transparent target emulation using QEMU user-mode execution.

Targets and naming

Target name precisely defines the target OS, architecture and the intended way of usage (“native” or cross-compilation).

Naming convention: OS-TYPE-ARCH

(e.g.: `linux-native-riscv64`, `linux-cross-mips`)

Linux targets:

- x86 (i386, amd64)
- ARM (armel/ARMv5, armhf/ARMv7, arm64/AARCH64)
- RISC-V (riscv64)
- MIPS (mips, mipsel, mips64el)
- PowerPC (ppc, ppc64, ppc64le)

Windows (cross-compilation) targets:

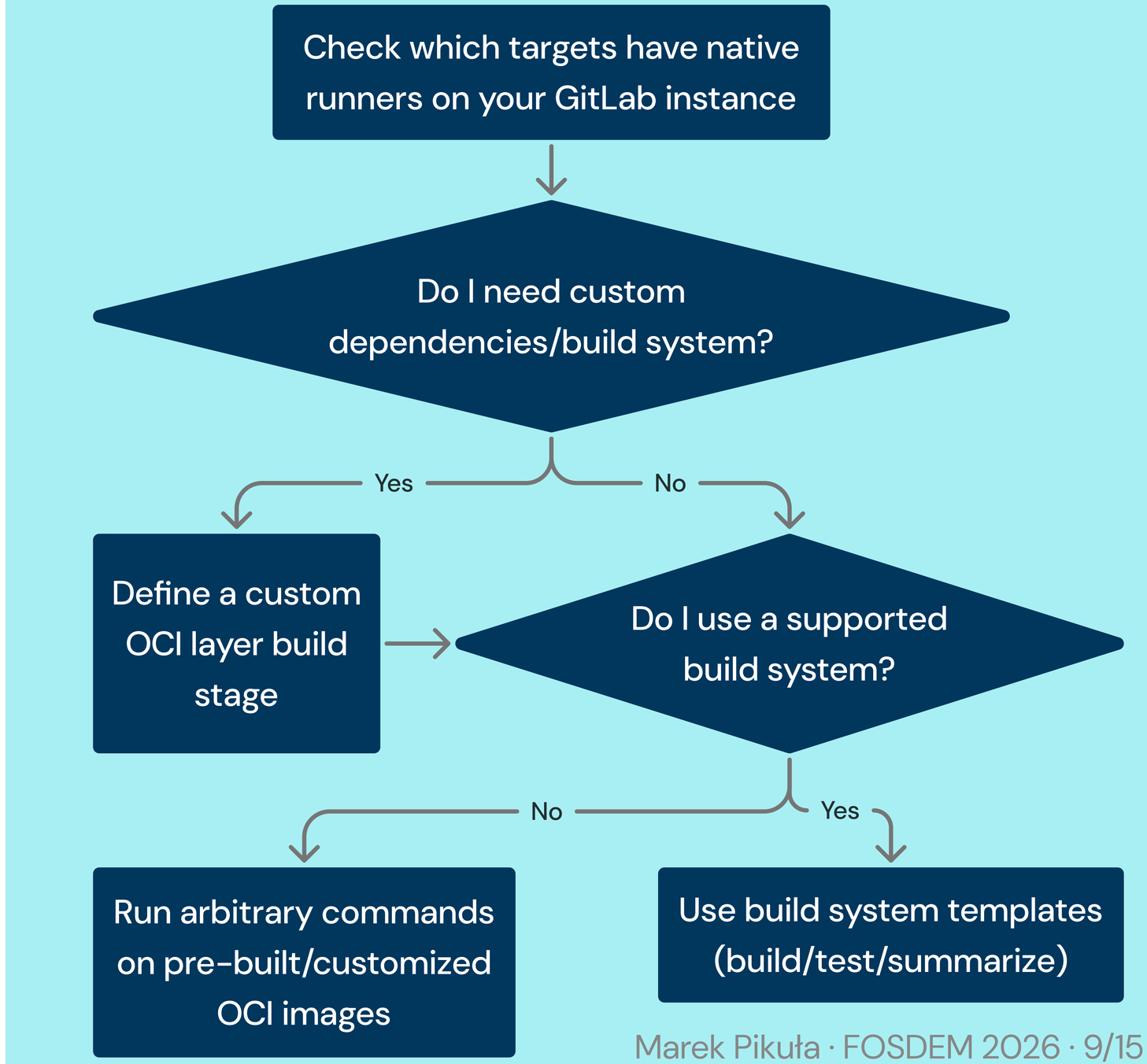
- x86 (i686, amd64)
- ARM64 (LLVM-only)

How to use it?

Templates expand a target matrix into per-architecture jobs that run in layered OCI images (native or via QEMU user-mode) and publish results.

```yaml
spec:
  inputs:
    runner_tag_amd64:
    runner_tag_qemu:
    runner_tag_arm64:
---

test:
  stage: test
  extends: .ci-multiplatform-base
  needs: [oci]
  tags: ["$RUNNER_TAG"]
  parallel:
    matrix:
      - TARGET: linux-native-amd64
        RUNNER_TAG: $[[ inputs.runner_tag_amd64 ]]
      - TARGET: linux-native-riscv64
        RUNNER_TAG: $[[ inputs.runner_tag_qemu ]]
      - TARGET: linux-native-arm64-v8
        RUNNER_TAG: $[[ inputs.runner_tag_arm64 ]]
  script:
    - uname -a
    - hello
```



Introduction

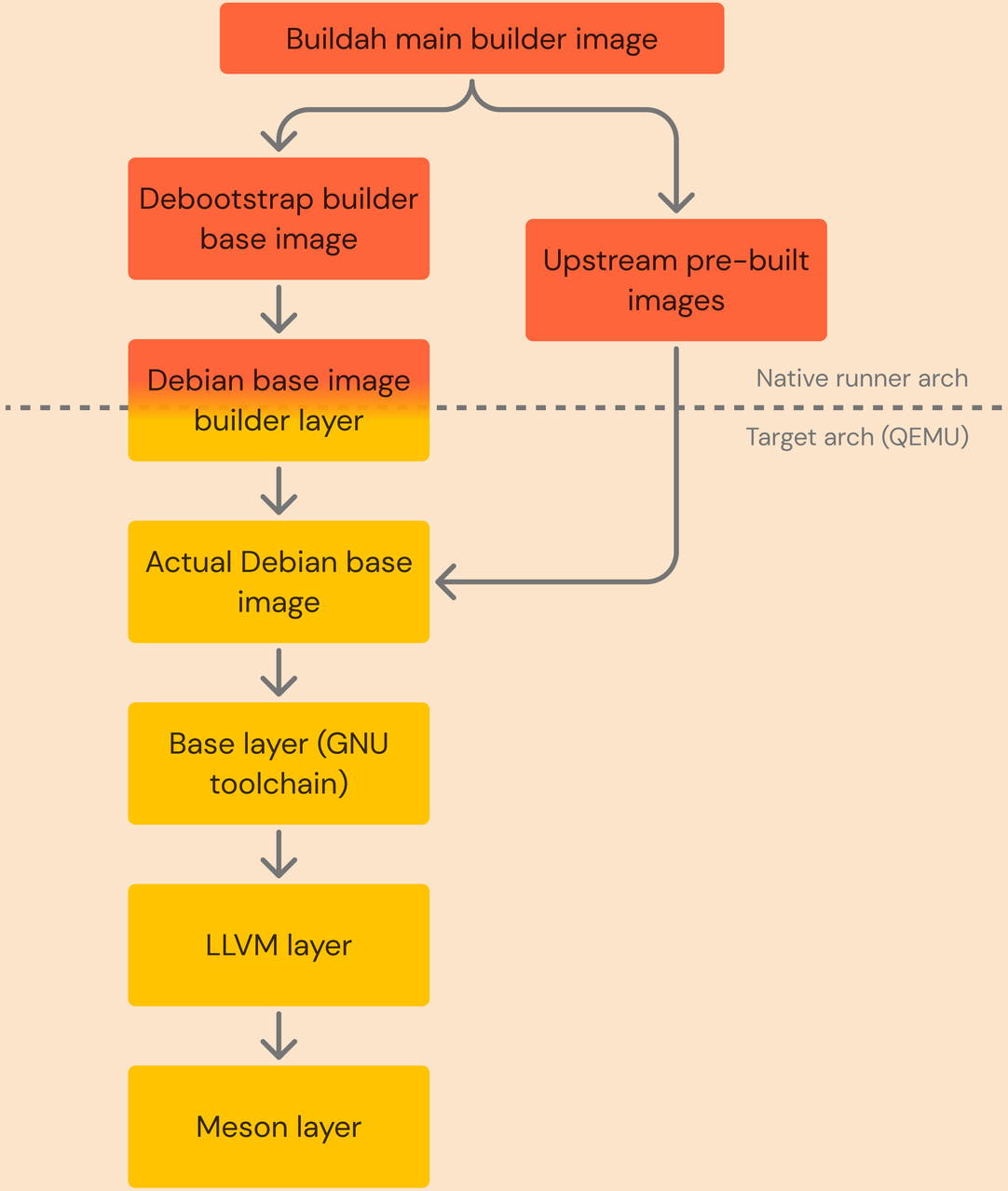
What it is?

→ **Hot takes**

What next?

Debian base images

- Custom Buildah base image – no official RISC-V Fedora images
- Debootstrap + Debuerreotype + Proot for Debian Ports
- Aggressive cache of build stages
- Upstream pre-built images where possible
- Optional, stacked layers:
  - GNU toolchain
  - LLVM toolchain
  - Meson + gcovr
  - (other build systems planned)



Virtual target profiles

- Test **virtual hardware profiles** without owning hardware, including configurations that are not yet available in production.
- **Customize `QEMU_CPU`** to select specific virtual CPU features for a target.
- As easy as extending the **parallel matrix** in GitLab CI to run the same tests across multiple virtual targets.
- It **extends platform coverage** beyond what is readily available, so the codebase is already validated when the ecosystem catches up.

```yaml
spec:
  inputs:
    ci_path:
    runner_tag_amd64:
    runner_tag_qemu:
    runner_tag_arm64:
---

include:
- component: $[[ inputs.ci_path ]]
  inputs:
    target: linux-native-riscv64
    runner_tag: $[[ inputs.runner_tag_qemu ]]
    parallel_matrix:
    - TOOLCHAIN: [gnu, llvm]
      QEMU_CPU:
      - rv64,v=false
      - rv64,v=true,vext_spec=v1.0,vlen=128,elen=64
      - rv64,v=true,vext_spec=v1.0,vlen=256,elen=64
      - rv64,v=true,vext_spec=v1.0,vlen=512,elen=64
      - rv64,v=true,vext_spec=v1.0,vlen=1024,elen=64
```

RISC-V sweep across VLEN values for both GNU and LLVM builds.

Introduction

What it is?

Hot takes

→ **What's next?**

WHAT'S NEXT?

What's next?

① Test in your project!

Check if you could benefit from testing your project across multiple architectures.

② Feedback and bugs

If you have any feedback or bugs after testing in your project don't hesitate to submit it on GitLab!

③ More build tools in the pipeline

Next milestone is adding integrated CMake and Autotools templates besides the existing Meson.

Thank you!

Questions?

https://gitlab.com/riseproject/ci/ci-multiplatform

m.pikula@partner.samsung.com

linkedin.com/in/marek-pikula

github.com/MarekPikula