If your ONLY goal is to compile your Aya Rust eBPF crate (lock-ebpf) to a working eBPF ELF binary for bpfel-unknown-none, then:

There are only a few things left to "look at" or perfect, and you already have most of them done.

I'll list out what you *could* want to double-check or improve next, and what you can safely ignore:

---

1. Absolute Minimum Checklist

- **bpf-linker** installed, and your eBPF crate (lock-ebpf) is configured to use it.
- **rust-toolchain.toml** in repo root, with [toolchain] channel = "nightly" and targets = ["bpfel-unknown-none"]
- **.cargo/config.toml** for lock-ebpf crate (not workspace root) with linker/rustflags (as above).
- **xtask** or script runs cargo +nightly build --target bpfel-unknown-none -Z build-std=core,alloc --release from lock-ebpf directory.
- **#![no_std] and #![no_main]** are present in your eBPF main.rs.
- **Aya** crate version matches what's expected for your code.
- **Check that target/bpfel-unknown-none/release/lock ELF is generated.**
- **Your loader and rest of the workspace are built for x86_64-unknown-linux-gnu** (not for BPF target).

---

2. Advanced / "It Just Works" Checks

- **Does the binary say "eBPF ELF"?**
    - Run file kernel/aya/target/bpfel-unknown-none/release/lock
    - Output should mention "eBPF", not "LLVM bitcode"
- **Try loading with bpftool** (if you want to be really sure).
- **BTF sections present?**
    - If you need portability (CO-RE), add -C link-arg=--btf as a rustflag.

- **No “std” in dependencies:**
    - eBPF programs can’t use std, only core/alloc/crates marked no_std.
    - Your lock-ebpf/Cargo.toml is good.

---

3. Extra (If you want maximum success):

- **Build and run a minimal “hello world” eBPF program with your pipeline** (to prove the build is sane even before loading system call probes).
- **Ensure Docker container has enough memory/disk (at least 10-15GB free).**
- **Clean up the build cache before each test run** (cargo clean, docker system prune).
- **Try building on Linux natively** (if Docker/Windows gets too flaky).
- **If you hit “Force-skipping unavailable component 'rust-std-bpfel-unknown-none'”**, it’s OK as long as you’re using -Z build-std=core,alloc.

---

4. You do NOT need to:

- Worry about compiling for “all kernels”.
- Add any stdlib shims.
- Touch anything in the workspace root’s .cargo/config if you want sane multi-target builds.
- Mix BPF flags into your main application crate.

---

5. If it still fails to produce a valid eBPF ELF binary:

- Double-check .cargo/config.toml **location** (must be in lock-ebpf only!).
- Try running cargo clean and build again.
- Reinstall bpf-linker in the container if you changed anything (cargo install bpf-linker --force).

So, What Else?

- **If you pass the checklist above, you're done.**
- If not, tell me *exactly* what's missing:
    - Are you NOT getting an ELF eBPF binary?
    - Does it say "LLVM bitcode"?
    - Does Aya fail to load the program?
    - Are there errors/warnings about the linker, or std?

Just let me know which checklist item you want to go deeper on, or if you want to try building from scratch and verify the file output.
If you want, I'll walk you through a "hello world" eBPF Rust/Aya from total zero, step by step, with example code and config.