Linux IoT Development: Adjusting from a Binary OS to the Yocto Project Workflow

Introducing the Yocto Project and the benefits of using it in embedded Linux development.

By Mirza Krak

In embedded Linux development, there are two approaches when it comes to what operating system to run on your device. You either build your own distribution (with tools such as Yocto/OpenEmbedded-Core, Buildroot and so on), or you use a binary distribution where Debian and derivatives are common.

It's common to start out with a binary distribution. This is a natural approach, because it's a familiar environment for most people who have used Linux on a PC. All the commodities are in place, and someone else has created the distribution image for you to download. There normally are custom vendor images for specific hardware that contain optimizations to make it easy to get started to utilize your hardware fully.

Any package imaginable is an `apt install` command away. This, of course, makes it suitable for prototyping and evaluation, giving you a head start in developing your application and your product. In some cases, you even might ship pre-series devices using this setup to evaluate your idea and product further. This is referred to as the "golden image" approach and involves the

following steps:

1. Flash the downloaded Debian image to an SD card.

2. Boot the SD card, log in and make any modifications needed (for example, installing custom applications). Once all the modifications are complete, this becomes your golden image.

3. Duplicate the SD card into an image on your workstation (for example, using `dd`).

4. Flash the "golden image" to a fleet of devices.

And every time you need to make a change, you just repeat steps 2–4, with one change—that is, you boot the already saved "golden image" in step 2 instead of the "vanilla" image.

At a certain point, the approach of downloading a pre-built distribution image and applying changes to it manually will become a problem, as it does not scale well and is error-prone due to the amount of manual labor that can lead to inconsistent output. The optimization would be to find ways to automate this, generating distribution images that contain your applications and your configuration in a reproducible way.

This is a crossroad where you decide either to stick with a binary distribution or move your idea and the result of the evaluation and prototyping phase to a tool that's able to generate custom distributions and images in a reproducible and automated way.

There are, of course, ways to generate custom Debian images, but the problem here is fragmentation. If you're using vendor-provided images, they probably have created their own tools (a bash script wrapper around debootstrap) to generate those images that you might be able to get access to. The fragmentation results

in very little re-use, and if you decide to change the hardware later but still base it on Debian, you might need to re-work your process completely, as this might be using a different set of tools.

The remainder of this article assumes you've made the choice to switch to a tool that's able to build Linux distributions—specifically the Yocto Project, which is based on OpenEmbedded-Core. This has some implications, and I try to cover the key parts.

Here's a quote from the yoctoproject.org website to give you a quick summary of the Yocto Project:

> The Yocto Project (YP) is an open-source collaboration project that helps developers create custom Linux-based systems regardless of the hardware architecture.
>
> The project provides a flexible set of tools and a space where embedded developers worldwide can share technologies, software stacks, configurations, and best practices that can be used to create tailored Linux images for embedded and IOT devices, or anywhere a customized Linux OS is needed.

Next, let's look at the key differences when moving from a binary distribution to the Yocto Project that will impact your workflow.

The Yocto Project Is a Cross-Development Environment

Because the Yocto Project is a cross-development environment, this means the build and generation of the custom Linux distribution image happens on the host machine, with the intention of the output running on a target. This could mean that the host is an x86_64 machine and the target is an ARM-v7—hence, the "cross-development".

And, this can be frightening at first; coming from a binary distribution, you might

not have encountered this workflow before. In the "golden image" example, you perform all the changes on the actual devices, but that's rarely needed in a Yocto environment where changes are applied during the build on the host machine, and you only provision your target device with the output image.

You can read more about cross-compilers on this Wikipedia post. It's important to build an understanding of this concept for a more seamless experience with Yocto.

The Yocto Project Is a Ground-Up Approach

The starting point in Yocto is to build a distribution image that contains the necessities to boot a system, and that is about it. This is, of course, not very useful on its own, but it's the foundation that you build upon and where only the components that you select to be included are included, and nothing else. This means you're in full control of the distribution that's generated, and it can be tailored for very specific use cases.

Understanding which components make up a Linux distribution might require additional knowledge acquisition. It's not normally something that you piece together when working with binary distributions, as it's already done by someone else. A good reference is the Linux From Scratch project, which is a step-by-step tutorial on how to create your own Linux distribution. It's essentially what Yocto does but in an automated fashion. I don't believe that you need to understand each step that's involved in detail, but you can use the LFS book to get a quick overview of which components can make up a Linux distribution.

The Yocto Project Builds from Source Code

This means instead of working with binary packages, which have been compiled and packaged by someone else, with the Yocto Project, you work with metadata that describes how to build packages from source code. This implies that everything is built from source, including toolchains, Linux kernel images, bootloader images, applications and more.

The workflow in Yocto to install a package onto your custom distribution is

to change the configuration and rebuild. This is the equivalent of running `apt install` on a Debian distribution.

The "build from source" approach is one of the Yocto Project's strengths in the flexibility it provides. Using this approach, you can customize every single package to your needs, and all the changes necessary are applied at build time, which means the output image will contain all the desired customizations—that is a big difference compared to working with a "golden image".

There also are drawbacks to the "build from source" approach. It has significant impact on the time it takes to construct a distribution image, which typically ranges in hours in build time, and involves steps such as fetching source code, unpacking, compiling, installing and so on. Yocto does support a caching mechanism, meaning that subsequent builds will be much faster and are typically in the range of minutes instead of hours.

Because Yocto is a resource-heavy tool, a project using it needs to plan for infrastructure changes and optimizations. This could involve sourcing capable machines to speed up builds or setting up build servers that can be utilized by developers but also where one could run automated builds. There are many optimizations that can be done within Yocto to help speed up builds; you can read more about it in the Yocto Project Mega-Manual.

The Yocto Project—Open-Source License Compliance

It's worth mentioning that open-source license compliance is slightly different in Yocto, because you hold the sources of the produced binaries, in case you need to re-distribute it—for example, for GPL-licensed code. In a binary distribution, the source code of the produced binaries are hosted by the distribution.

For more information, see the section Maintaining Open Source License Compliance During Your Product's Lifecycle in the Yocto Mega-Manual.

Conclusion

Yocto has a steep learning curve, and you should be prepared for it. Yocto has its strength in flexibility, but this also adds to the complexity.

It's fairly easy to do a quick build of a base image, but the hurdle is often moving from this to creating something custom, and to do this, you'll need to spend time reading the Yocto Project Mega-Manual to understand the core concepts. Here are a few topics to start with:

- The layer structuring and overlays, which includes core layers, board support package layers and distribution layers.
- The bitbake build engine and how it parses metadata and executes tasks.
- The Yocto Project workflow.

The investment of learning Yocto does come with benefits that will be valuable in the long-term:

- You will have mastered a tool that gives full insights and control on how you are building the distribution image. The deeper understanding of the system definitely will be helpful when debugging problems once products are in the field.
- You will have a streamlined development workflow with higher automation and a higher level of reproducibility.
- You will have increased re-usability of the software stack if you decide to change hardware or plan to release similar products that are based on the same platform.
- You will gain access to a large community of experienced developers that are

willing to help you along the way.

- You will gain access to a large ecosystem of production-grade software that is easily integrated into your software stack due to Yocto's layering design. ■

Mirza Krak is an embedded Linux solution specialist with the open-source Mender.io project to manage updates for IoT. He has eight years of experience in the field. He is involved in various other open-source projects and is a Linux kernel contributor. Mirza has spoken at various conferences including the Embedded Linux Conferences.

Resources

A large portion of the information and reflections in this article comes from my experience working in the embedded Linux field and my involvement in numerous projects that shipped products based on Linux.

Further resources:

- yoctoproject.org
- Yocto Project Mega-Manual
- Yocto Project Wiki

Send comments or feedback
via http://www.linuxjournal.com/contact
or email ljeditor@linuxjournal.com.